Rational Rhapsody Property Definitions

Generated on April 26, 2009

Table of Contents

Activity_diagram 15
- AcceptEventAction 16
- Action 17
- ActionBlock 18
- ActionPin 19
- ActivityParameter 20
- AutoPopulate 22
- ButtonArray 23
- CallOperation 24
- Comment 25
- Complete 30
- DecisionNode 31
- ConnectorText 31
- Constraint 32
- DefaultTransition 36
- DependentText 39
- Depends 40
- DiagramConnector 43
- DigitalDisplay 44
- Gauge 44
- General 45
- HistoryConnector 46
- JunctionConnector 46
- Knob 47
- Labels 47
- Led 48
- LevelIndicator 48
- LoopTransition 48
- MatrixDisplay 49
- Meter 50
- Names 50
- Note 51
- ObjectFlowState 52
- ObjectNode 53
- OnOffSwitch 54
- Partition 54
- PartitionFrame 55
- PushButton 55
- ReferenceActivity 56
- Requirement 59
- Requirement 63
- SelectorConnector 64
- SendAction 64
- Slider 65
- State 66
- StateDiagram 67

SubActivityState 67
Swimlane 70
TerminationConnector 71
TextBox 72
Transition 72
Ada_CG 76
Argument 76
Attribute 78
Class 85
Component 101
Configuration 101
Dependency 109
Event 113
File 115
Framework 122
Generalization 134
GNAT 135
GNATVxWorks 154
INTEGRITY 163
INTEGRITY5 167
Multi4Win32 172
MultiWin32 177
OBJECTADA 181
Operation 185
Package 194
Port 200
RAVEN_PPC 201
Relation 205
SPARK 214
Type 218
Ada_ReverseEngineering 224
ApproximatedConstructs 224
Filtering 225
ImplementationTrait 226
Main 229
MFC 230
MSVC60 231
Parser 232
Promotions 232
ADA_Roundtrip 234
General 234
CPP_Roundtrip::Type 236
Update 237
Animation 238
ClassifierRole 238
ATL 240
Class 240
Configuration 241
Macro 246
Operation 251

Browser 253
Operation 253
Settings 253
CG 257
Argument 257
Attribute 258
CGGeneral 261
Class 262
Component 269
Configuration 271
Dependency 276
Event 278
File 282
General 286
Generalization 287
Operation 288
Package 290
Relation 294
Statechart 303
Type 305
Collaboration_Diagram 308
AssociationRole 308
AutoPopulate 309
Classifier 309
ClassifierActor 310
CollaborationDiagramGE 310
CollMessage 310
Comment 311
Complete 312
Constraint 312
Depends 313
Messages 314
MultiObj 314
Note 314
Requirement 315
ReverseCollMessage 316
COM 317
Argument 317
Attribute 319
Class 322
coclass 323
Configuration 325
IDL 327
Interface 329
Library 332
Operation 334
Relation 336
ComponentDiagram 337
AutoPopulate 337
Class 337

Comment 339
Complete 340
Component 340
ComponentDiagramGE 341
Constraint 342
Depends 342
FileComponent 343
Flow 344
FolderComponent 345
InterfaceComponent 346
Note 347
CompRealization 347
Requirement 349
ConfigurationManagement 350
ClearCase 350
General 368
PVCS 373
SCC 386
SourceIntegrity 394
Synergy 404
CORBA 407
C++Mapping_CORBABasic 407
C++Mapping_CORBAEnum 408
C++Mapping_CORBAFixedArray 409
C++Mapping_CORBAFixedSequence 410
C++Mapping_CORBAFixedStruct 411
C++Mapping_CORBAFixedUnion 412
C++Mapping_CORBAInterfaceReference 413
C++Mapping_CORBAInterfaceVariable 414
C++Mapping_CORBASequence 415
C++Mapping_CORBAVariableArray 416
C++Mapping_CORBAVariableStruct 417
C++Mapping_CORBAVariableUnion 418
Class 419
Configuration 421
Operation 422
Package 423
TAO 423
Type 430
UserDefinedORB 431
CPP_CG 439
Argument 440
Attribute 442
Class 452
Configuration 468
Cygwin 476
Dependency 492
Event 496
File 499
Framework 506

General 520
Generalization 520
INTEGRITY 521
INTEGRITY5 531
Integrity5ESTL 548
IntegrityESTL 565
Linux 581
Microsoft 592
MicrosoftDLL 605
MicrosoftWinCE600 619
MSStandardLibrary 632
Multi4Linux 645
Multi4Win32 654
MultiWin32 670
NucleusPLUS-PPC 681
Operation 692
OsePPCDiab 702
OseSfk 715
Package 729
Port 735
QNXNeutrinoMomentics 737
QNXNeutrinoGCC 750
Relation 762
Solaris2 771
Solaris2GNU 782
Statechart 793
Type 794
VxWorks 798
VxWorks6diab 811
VxWorks6diab_RTP 824
VxWorks6gnu 837
VxWorks6gnu_RTP 850
WorkbenchManaged 863
WorkbenchManaged_RTP 875
CPP_ReverseEngineering 887
Filtering 887
ImplementationTrait 888
Main 897
MFC 898
MSVC60 898
Parser 899
Promotions 901
Update 902
CPP_Roundtrip 905
General 905
Update 909
C_CG 910
Argument 911
Attribute 913
Class 922

Configuration 938
Cygwin 948
Dependency 964
Event 967
File 971
Framework 979
Generalization 992
INTEGRITY 993
INTEGRITY5 1003
Link 1018
Linux 1019
Microsoft 1032
MicrosoftIDF 1045
ModelElement 1056
Multi4Win32 1059
NucleusPLUS-PPC 1078
Operation 1090
Package 1101
Port 1107
Relation 1107
Solaris2 1117
Solaris2GNU 1129
Statechart 1140
Type 1142
VxWorks 1147
VxWorks6diab 1160
VxWorks6diab_RTP 1174
VxWorks6gnu 1187
VxWorks6gnu_RTP 1201
WorkbenchManaged 1214
WorkbenchManaged_RTP 1227
C_ReverseEngineering 1241
Filtering 1241
ImplementationTrait 1242
Main 1251
MFC 1252
MSVC60 1252
Parser 1253
Promotions 1255
Update 1256
C_Roundtrip 1259
General 1259
Update 1261
DeploymentDiagram 1263
AutoPopulate 1263
Comment 1263
Communication_Path 1264
Complete 1265
ComponentInstance 1265
Constraint 1266

Depends 1267
DeploymentDiagramGE 1268
Flow 1268
NodeProcessor 1269
Note 1270
Requirement 1270
DiagramPrintSettings 1272
General 1272
Dialog 1274
All 1274
Attribute 1274
Class 1275
ClassifierRole 1276
Component 1276
Configuration 1277
Dependency 1277
Diagrams 1278
Event 1278
File 1278
General 1279
ObjectModelDiagram 1280
Operation 1280
Package 1281
Project 1282
Relation 1283
SequenceDiagram 1283
Stereotype 1284
Type 1284
UseCaseDiagram 1284
Eclipse 1286
Configuration 1286
DefaultEnvironments 1287
Export 1287
General 1289
Graphics 1289
Model 1299
ModelLibraries 1318
Profile 1318
Relations 1318
Report 1319
ReporterPLUS 1319
Workspace 1321
IntelliVisor 1324
General 1324
PredefineMacros 1325
PredefineMacrosTooltip 1326
Java(1.1)Containers 1327
BoundedOrdered 1327
BoundedUnordered 1334
Fixed 1340

General 1347
Qualified 1347
Scalar 1354
StaticArray 1356
UnboundedOrdered 1362
UnboundedUnordered 1366
User 1371
Java(1.2)Containers 1377
BoundedOrdered 1377
BoundedUnordered 1385
Fixed 1393
General 1400
Qualified 1401
Scalar 1409
StaticArray 1416
UnboundedOrdered 1424
UnboundedUnordered 1432
User 1440
Java(1.5)Containers 1448
BoundedOrdered 1448
BoundedUnordered 1456
Fixed 1464
General 1471
Qualified 1472
Scalar 1480
StaticArray 1487
UnboundedOrdered 1495
UnboundedUnordered 1503
User 1511
JAVA_CG 1519
AnimInstrumentation 1519
Argument 1519
Attribute 1520
Class 1528
Component 1542
Configuration 1542
Dependency 1550
Event 1553
File 1555
Framework 1562
Generalization 1574
JDK 1575
Operation 1586
Package 1594
Port 1599
Relation 1599
Statechart 1608
Type 1608
JAVA_ReverseEngineering 1613
Filtering 1613

ImplementationTrait 1614
Main 1621
MFC 1622
MSVC60 1622
Parser 1623
Promotions 1624
JAVA_Roundtrip 1626
General 1626
Type 1628
Update 1629
Model 1630
Attribute 1630
Class 1631
ControlledFile 1632
MatrixLayout 1634
MatrixView 1634
Profile 1635
Stereotype 1636
TableLayout 1642
Type 1643
ObjectModelGe 1644
Actor 1644
Aggregation 1645
Association 1648
Attribute 1650
AutoPopulate 1651
Class 1652
ClassDiagram 1653
Comment 1654
Complete 1656
Composition 1657
Constraint 1659
ContainArrow 1661
Depends 1662
Flow 1663
Inheritance 1664
Link 1665
Note 1668
Object 1668
Package 1670
PrimitiveOperation 1671
Requirement 1672
Stereotype 1674
Tag 1674
Type 1675
UseCase 1676
OMContainers 1678
BoundedOrdered 1678
BoundedUnordered 1685
EmbeddedFixed 1692

EmbeddedScalar 1699
Fixed 1705
General 1712
Qualified 1712
Scalar 1719
StaticArray 1726
UnboundedOrdered 1733
UnboundedUnordered 1740
User 1747
OMCorba2CorbaContainers 1754
BoundedOrdered 1754
BoundedUnordered 1762
EmbeddedFixed 1770
EmbeddedScalar 1777
Fixed 1785
General 1793
Qualified 1793
Scalar 1801
StaticArray 1808
UnboundedOrdered 1816
UnboundedUnordered 1824
User 1831
OMCpp2CorbaContainers 1840
BoundedOrdered 1840
BoundedUnordered 1848
EmbeddedFixed 1856
EmbeddedScalar 1863
Fixed 1871
General 1879
Qualified 1879
Scalar 1887
StaticArray 1894
UnboundedOrdered 1902
UnboundedUnordered 1910
User 1918
OMCppOfCorbaContainers 1926
BoundedOrdered 1926
BoundedUnordered 1934
EmbeddedFixed 1942
EmbeddedScalar 1949
Fixed 1957
General 1965
Qualified 1965
Scalar 1973
StaticArray 1980
UnboundedOrdered 1988
UnboundedUnordered 1996
User 2004
OMUContainers 2012
BoundedOrdered 2013

BoundedUnordered 2020
EmbeddedFixed 2027
EmbeddedScalar 2034
Fixed 2042
General 2049
Qualified 2049
Scalar 2057
StaticArray 2064
UnboundedOrdered 2071
UnboundedUnordered 2079
User 2086
PanelDiagram 2094
ButtonArray 2094
DigitalDisplay 2095
Gauge 2095
General 2096
Knob 2096
Led 2097
LevelIndicator 2097
MatrixDisplay 2098
Meter 2098
OnOffSwitch 2099
PushButton 2099
Slider 2100
TextBox 2101
QoS 2102
Class 2102
Operation 2104
Resource 2104
ReverseEngineering 2106
Main 2106
Progress 2107
Update 2109
RiCContainers 2111
BoundedOrdered 2111
BoundedUnordered 2119
EmbeddedFixed 2127
EmbeddedScalar 2134
Fixed 2142
General 2150
Qualified 2150
Scalar 2158
StaticArray 2166
UnboundedOrdered 2174
UnboundedUnordered 2181
User 2189
RoseInterface 2198
Import 2198
RTInterface 2200
DOORS 2200

ExportOptions 2201
SequenceDiagram 2210
Condition_Mark 2210
General 2210
InstanceLine 2215
InteractionOperator 2215
Message 2215
SequenceDiagram 2216
SPARK 2217
Class 2217
Package 2217
StatechartDiagram 2219
AutoPopulate 2219
ButtonArray 2219
Comment 2220
Complete 2222
CompState 2222
Constraint 2223
DefaultTransition 2225
Depends 2226
DigitalDisplay 2227
Gauge 2227
General 2228
Knob 2228
Led 2229
LevelIndicator 2229
MatrixDisplay 2230
Meter 2230
Note 2231
OnOffSwitch 2231
PushButton 2232
Requirement 2232
SendAction 2234
Slider 2235
State 2236
StateDiagram 2237
TextBox 2238
Transition 2238
STLContainers 2240
BoundedOrdered 2240
BoundedUnordered 2246
EmbeddedFixed 2253
EmbeddedScalar 2259
Fixed 2263
General 2270
Qualified 2270
Scalar 2277
StaticArray 2283
UnboundedOrdered 2289
UnboundedUnordered 2296

User 2302
TestConductor 2309
SDInstance 2309
SequenceDiagram 2309
Settings 2310
TestCase 2311
UseCaseExtensions 2314
Dependency 2314
UseCaseGe 2315
Actor 2315
Association 2316
AutoPopulate 2318
Comment 2319
Complete 2321
Constraint 2321
Depends 2323
Flow 2324
Inheritance 2326
Note 2327
Package 2327
Requirement 2328
SystemBox 2330
UseCase 2330
UseCaseDiagram 2332
WebComponents 2333
Attribute 2333
Class 2333
Configuration 2334
Event 2335
File 2336
Operation 2336
WebFramework 2336
WSDL 2338
Package 2338
XSD 2339
Type 2339

Activity_diagram

The subject Activity_diagram contains the following metaclasses with properties for controlling the activity diagram editor. Any addition metaclasses, listed below, that do not appear in the Features window are included in the definitions for backwards compatibility with previous Rational Rhapsody versions.

- AcceptEventAction
- Action
- ActionBlock
- ActionPin
- ActivityParameter
- AutoPopulate
- CallOperation
- Comment
- Complete
- ConnectorText
- Constraint
- DecisionNode
- DefaultTransition
- Depends
- DiagramConnector
- General
- HistoryConnector
- JunctionConnector
- Labels
- LoopTransition
- Names
- Note
- ReferenceActivity
- Requirement
- ObjectFlowState
- Partition
- PartitionFrame
- SendAction
- ShowStereotype
- State
- SelectorConnector
- StateDiagram
- SubActivityState

- Swimlane
- TerminationConnector
- Transition

AcceptEventAction

The metaclass AcceptEventAction contains properties that affect the appearance of accept event actions in activity diagrams.

ShowNotation

The property ShowNotation determines what text is opened on Accept Event Action elements in an activity diagram. The possible values are:

- Name - The name of the element is displayed.
- Label - The label of the element is displayed.
- Event - The name of the event selected is displayed.
- FullNotation - In addition to the name of the event selected, Rational Rhapsody displays the name of the target selected and the argument values you provided.

Note: This property can only be set at the diagram level or higher and not at the level of individual Accept Event Action elements.

When you change the value of this property, the display of any new Accept Event Action elements are affected, but the display of Accept Event Action elements already on the diagram remains as is.

(Default = FullNotation)

ShowStereotype

The ShowStereotype property determines if, and how, an stereotypes of the element are displayed in a diagram. The possible values are:

- Label - The stereotypes of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is displayed.
- None - The stereotypes of the element are not displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Accept Event Action elements).

When you change the value of this property, the display of any new Accept Event Action elements are affected, but the display of Accept Event Action elements already on the diagram remains as is.

(Default = None)

Action

The Action metaclass contains properites to control the appearance of actions in activity diagrams.

ShowAction

The ShowAction property controls what is displayed inside the action block. The possible values are as follows:

- Action - Display the name of the action
- Description - Display the description of the action
- Label - Display the label of the action

(Default = Action)

showName

The showName property specifies how the name of an object should be displayed.

The possible values are as follows:

- Name - Display the name of the action.
- Label - Display the label of the action.
- None - Display neither the name nor label for the action.

(Default = None)

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition

- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

ActionBlock

The ActionBlock metaclass contains a property to control the appearance of action blocks in activity diagrams.

showName

The showName property specifies how the name of an object should be displayed.

The possible values are as follows:

- Name - Display the name of the action.
- Label - Display the label of the action.
- None - Display neither the name nor label for the action.

(Default = None)

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition

- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

ActionPin

The ActionPin metaclass contains properties that control the appearance of action pins in activity diagrams.

showName

The property showName determines what text is opened alongside Action Pin elements. The possible values are:

- Name - The name of the element is displayed.
- NameAndType - Both the name and the type (e.g., int) are displayed.
- Type - The type (e.g., int) is displayed.
- Label - The label of the element is displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Action Pin elements).

When you change the value of this property, the display of any new Action Pin elements are affected, but the display of Action Pin elements already on the diagram remains as is.

Default = Name

ActivityParameter

The ActivityParameter metaclass contains properties determine how to display the name and stereotype.

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition

• Activity_diagram::DefaultTransition
• Activity_diagram::ReferenceActivity
• Activity_diagram::SubActivityState
• Activity_diagram::ActivityParameter
• Name, Label, None (default is None)
• Used for following subject::metaclass combinations:
• ObjectModelGe::Inheritance
• ObjectModelGe::Depends
• UseCaseGe::Depends
• UseCaseGe::Inheritance
• Activity_diagram::ActivityParameter
• Activity_diagram::Depends
• Statechart::Depends
• Full_path, Relative, Name_only, Label (default is Relative)
• Used for following subject::metaclass combinations:
• Statechart::Requirement
• ObjectModelGe::Class
• ObjectModelGe::Object
• ObjectModelGe::UseCase
• ObjectModelGe::Actor
• ObjectModelGe::Requirement
• UseCaseGe::Actor
• UseCaseGe::Association
• UseCaseGe::Requirement
• UseCaseGe::UseCase
• Activity_diagram::Requirement
• Name, Description, Label (default is Description)
• Used for following subject::metaclass combinations:
• Statechart::Comment
• ObjectModelGe::Comment
• UseCaseGe::Comment
• Activity_diagram::Comment
• Name, Specification, Description, Label (default is Description)
• Used for following subject::metaclass combinations:
• Statechart::Constraint
• ObjectModelGe::Constraint
• UseCaseGe::Constraint
• Activity_diagram::Constraint

- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rational Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rational Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rational Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Top-Bottom

LayoutStyle

The property LayoutStyle is used when Rational Rhapsody automatically generates a diagram, and it determines the general appearance of the diagram - hierarchical or orthogonal.

- Hierarchical - diagram layout will reflect a hierarchy, appropriate for relationships such as inheritance
- Orthogonal - diagram layout will resemble a grid, appropriate where there are no clear hierarchical relationships between the elements in the diagram

There are two situations where Rational Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click in the browser a diagram that was generated using the Rational Rhapsody API.

Default = Hierarchical

ButtonArray

The ButtonArray metaclass contains properties that determine the appearance and behavior of button array controls on activity diagrams.

ButtonFont

The ButtonFont property lets you select the font to use for the text on the face of a push button control.

To change the value of the property, click the "..." button in the box next to the property value to open the Font window. The value of the property affects both buttons already on the activity diagram and new buttons added to the diagram. (The display of buttons already on the diagram changes only after you refresh the diagram.)

Default = Arial 10 NoBold NoItalic

Direction

The Direction property determines whether the button array controls are used to input data, display data, or both. The possible values are:

- In - The button arrays are only used to input data for the attribute to which it is bound.
- Out - The button arrays are only used to display data for the attribute to which it is bound.
- InOut - The button arrays are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for button array elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the button array.
- BindedElement - The name of the attribute that is bound to the button array.
- Name - The name of the button array element.
- None - No text is displayed.

Default = Name

CallOperation

The CallOperation metaclass contains properties that relate to Call Operation elements in activity diagrams.

ShowAction

The property ShowAction determines what is opened on Call Operation elements in an activity diagram. The possible values are:

- Action - The code entered in the Action text box is displayed.
- Description - The description entered for the element is displayed.
- Label - The label entered for the element is displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Call Operation elements). When you change the value of this property, the display of any new Call Operation elements are affected, but the display of Call Operation elements already on the diagram remains as is.

Default = Action

showName

The property showName determines what text is opened at the top of Call Operation elements. The possible values are:

- Name - The name of the element is displayed.
- Label - The label of the element is displayed.
- Operation - The operation that is being called is displayed.
- FullNotation - In addition to the operation name, the arguments and return value of the operationare displayed.
- None - Nothing is displayed at the top of the element.

Note that this property can only be set at the diagram level or higher (not at the level of individual Call Operation elements). When you change the value of this property, the display of any new Call Operation elements are affected, but the display of Call Operation elements already on the diagram remains as is.

Default = Operation

ShowStereotype

The ShowStereotype property determines if, and how, an stereotypes of the element are displayed in a diagram. The possible values are:

- Label - The stereotypes of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is displayed.
- None - The stereotypes of the element are not displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Call Operation elements). When you change the value of this property, the display of any new Call Operation elements are affected, but the display of Call Operation elements already on the diagram remains as is.

Default = Label

Comment

The Comment metaclass contains properties that control the appearance of comments in activity diagrams.

CommentNotation

The CommentNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Comment:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles. If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties dialog or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

(Default = 1)

name_color

The name_color property specifies the default color of names of graphical items.

(Default = 0,0,0)

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available

options:

- Name
- Description
- Label

ShowForm

Determines how note-like elements are displayed. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

The various combinations of possible values are as follows:

- Plain, Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- Statechart::Note
- ObjectModelGe::Requirement
- ObjectModelGe::Comment
- ObjectModelGe::Constraint
- UseCaseGe::Requirement
- UseCaseGe::Comment
- UseCaseGe::Constraint
- Activity_diagram::Requirement
- Activity_diagram::Comment
- Activity_diagram::Constraint
- Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Note
- UseCaseGe::Note
- Activity_diagram:Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The

different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter

- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase
- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

DecisionNode

The DecisionNode metaclass contains a property that controls the appearance of condition connectors in activity diagrams.

CaptionBehavior

The property CaptionBehavior provides you with a certain degree of flexibility in terms of displaying text for a conditional connector.

By default, the value of the property is set to Fixed. This means that the amount of text displayed depends upon the size of the connector. If you have a lot of text and want it all to be displayed, you have to enlarge the connector in order to enlarge the text display area.

You can get around this limitation by setting the value of the property to Floating. This provides you with separate controls that can be used to enlarge the text display area without affecting the size of the connector itself.

In addition to the different behavior in terms of the size of the text display area, the value of the property also affects the position of the text. When set to Fixed, the text will always be displayed at the center of the connector. When set to Floating, you can move the text anywhere you want relative to the position of the connector.

Default = Fixed

show_name

The show_name property specifies how to label condition connectors in activity diagrams. The possible values are Name, Label, and None.

(Default = Label)

ConnectorText

The ConnectorText metaclass contains properties that control the appearance of text inside connectors in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,0,0)

Constraint

The Constraint metaclass contains properties that control the appearance of constraints in activity diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties dialog or directly in the .prp file. Rather, you should use the Display Options for an element to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

ConstraintNotation

The ConstraintNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Constraint:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles. If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available options:

- Name
- Description
- Label

ShowForm

Determines how note-like elements are displayed. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text

- Note - Color background behind text
- Pushpin - Color background plus pin icon

The various combinations of possible values are as follows:

- Plain, Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- Statechart::Note
- ObjectModelGe::Requirement
- ObjectModelGe::Comment
- ObjectModelGe::Constraint
- UseCaseGe::Requirement
- UseCaseGe::Comment
- UseCaseGe::Constraint
- Activity_diagram::Requirement
- Activity_diagram::Comment
- Activity_diagram::Constraint
- Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Note
- UseCaseGe::Note
- Activity_diagram:Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside

the package in the diagram.

- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement

- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase
- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action

- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

DefaultTransition

The DefaultTransition metaclass controls the appearance of a default transition in an activity diagram.

line_style

The line_style property specifies the type of line used for a graphical item. The possible values are:

- straight_arrows - a straight line.
- rectilinear_arrows - rectilinear lines with right-angled corners placed at appropriate locations, depending on the start and end points of the line.
- spline_arrows - curved line without corners.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is straight_arrows:

- UseCaseGe::Inheritance
- Activity_diagram::DefaultTransition
- Statechart::Depends

For the following subject::metaclass combinations, the default value is spline_arrows:

- Statechart::DefaultTransition
- Activity_diagram::SubActivityState

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends

- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase
- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:

- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

DependentText

The DependentText metaclass contains a property that controls the appearance of dependent text in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,0,255)

Depends

The Depends metaclass contains properties that control the appearance of dependency relation lines in collaboration diagrams.

line_style

The line_style property specifies the type of line used for a graphical item. The possible values are as follows:

- straight_arrows - a straight line.
- rectilinear_arrows - rectilinear lines with right-angled corners placed at appropriate locations, depending on the start and end points of the line.
- spline_arrows - curved line without corners.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is straight_arrows:

- UseCaseGe::Inheritance
- Activity_diagram::DefaultTransition
- Statechart::Depends

For the following subject::metaclass combinations, the default value is spline_arrows:

- Statechart::DefaultTransition
- Activity_diagram::SubActivityState

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information

that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.

- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor

- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase
- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition

- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,64)

UseFillcolor

The UseFillcolor property is a Boolean value that specifies whether to use the fill color specified for that type of connector or port.

(Default = False)

DiagramConnector

The DiagramConnector metaclass contains properties that control the appearance of diagram connectors in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,255)

text_color

The text_color property specifies the default text color.

(Default = 255,255,255)

DigitalDisplay

The DigitalDisplay metaclass contains properties that determine the appearance and behavior of digital display controls on activity diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for digital display elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the digital display.
- BindedElement - The name of the attribute that is bound to the digital display.
- Name - The name of the digital display element.
- None - No text is displayed.

Default = Name

Gauge

The Gauge metaclass contains properties that determine the appearance and behavior of gauge controls on activity diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for gauge elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the gauge.
- BindedElement - The name of the attribute that is bound to the gauge.
- Name - The name of the gauge element.
- None - No text is displayed.

Default = Name

General

The General metaclass contains a property that controls the general behavior of activity diagrams.

DefaultMode

The Features dialog for activity diagrams allows you to specify that an activity diagram should be "Analysis Only". This means that the diagram is for modeling purposes only and no code should be generated for the diagram. Certain types of model elements can only be included in "Analysis Only" activity diagrams.

The property DefaultMode can be used to specify the default value for this setting. The property can take the following values:

- Design - New activity diagrams are created in Design mode by default, i.e., code is generated for the diagram
- Analysis - New activity diagrams are created in Analysis mode by default, i.e., code will not be generated for the diagram

Default = Design

DeleteConfirmation

The DeleteConfirmation property specifies whether confirmation is required before deleting a graphical element from the model. Note that this property does not apply to statechart elements, which have a separate DeleteConfirmation property.

The possible values are as follows:

- Always - Rational Rhapsody displays a confirmation dialog each time you try to delete an item from the model.
- Never - Confirmation is not required to delete an element.
- WhenNeeded - Asks for confirmation if there are references to the element.

(Default = Never)

ShowSwimlaneHeadings

The property ShowSwimlaneHeadings is used to specify that Rational Rhapsody should display swimlane headings at the top of an activity diagram. This is useful when you are working with long swimlanes where the top of the swimlane disappears when you scroll down.

For individual diagrams, you can control the display of these headings using the context menu items Show Swimlane Headings, Hide Swimlane Headings.

Default = Checked

HistoryConnector

The HistoryConnector metaclass contains properties that control the appearance of history connectors in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,255)

UseFillcolor

The UseFillcolor property is a Boolean value that specifies whether to use the fill color specified for that type of connector or port.

(Default = True)

JunctionConnector

The JunctionConnector metaclass contains properties that control the appearance of junction connectors in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,255)

UseFillcolor

The UseFillcolor property is a Boolean value that specifies whether to use the fill color specified for that type of connector or port.

(Default = True)

Knob

The Knob metaclass contains properties that determine the appearance and behavior of knob controls on activity diagrams.

Direction

The Direction property determines whether the knob controls are used to input data, display data, or both. The possible values are:

- In - The knobs are only used to input data for the attribute to which it is bound.
- Out - The knobs are only used to display data for the attribute to which it is bound.
- InOut - The knobs are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for knob elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the knob.
- BindedElement - The name of the attribute that is bound to the knob.
- Name - The name of the knob element.
- None - No text is displayed.

Default = Name

Labels

The Labels metaclass contains a property that controls the appearance of labels in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,0,255)

Led

The LED metaclass contains properties that determine the appearance and behavior of LED controls on activity diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for LED elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the LED.
- BindedElement - The name of the attribute that is bound to the LED.
- Name - The name of the LED element.
- None - No text is displayed.

Default = Name

LevelIndicator

The LevelIndicator metaclass contains properties that determine the appearance and behavior of level indicator controls on activity diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for level indicator elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the level indicator.
- Name - The name of the level indicator element.
- None - No text is displayed.

Default = Name

LoopTransition

The LoopTransition metaclass contains properties that control the appearance of loop transition lines in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,64)

label_color

The label_color property is currently unused.

Activity_Diagram::LoopTransition/Transition/0,127,255

Statechart::Transition,0,127,255

(Default = 0, 127, 255)

line_style

The line_style property specifies the default line style for a graphical item.

The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw rectilinear lines with right-angled corners depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

(Default = spline_arrows)

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams.

The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

(Default = None)

MatrixDisplay

The MatrixDisplay metaclass contains properties that determine the appearance and behavior of matrix display controls on activity diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for matrix display elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the matrix display.
- BindedElement - The name of the attribute that is bound to the matrix display.
- Name - The name of the matrix display element.
- None - No text is displayed.

Default = Name

Meter

The Meter metaclass contains properties that determine the appearance and behavior of meter controls on activity diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for meter elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the meter.
- BindedElement - The name of the attribute that is bound to the meter.
- Name - The name of the meter element.
- None - No text is displayed.

Default = Name

Names

The Names metaclass contains a property that controls the appearance of activity names in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,0,0)

Note

The Note metaclass contains properties that control the appearance of notes in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,64)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

(Default = 1)

name_color

The name_color property specifies the default color of names of graphical items.

(Default = 0,128,255)

ShowForm

Determines how note-like elements are displayed. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

The various combinations of possible values are as follows:

- Plain, Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- Statechart::Note
- ObjectModelGe::Requirement
- ObjectModelGe::Comment
- ObjectModelGe::Constraint
- UseCaseGe::Requirement
- UseCaseGe::Comment
- UseCaseGe::Constraint
- Activity_diagram::Requirement
- Activity_diagram::Comment
- Activity_diagram::Constraint
- Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Note
- UseCaseGe::Note
- Activity_diagram:Note

ObjectFlowState

The ObjectFlowState metaclass contains properties that control the appearance of object flow states in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,255,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action

state lines).

(Default = 1)

name_color

The name_color property specifies the default color of names of graphical items.

(Default = 0,0,0)

ObjectNode

The ObjectNode metaclass contains properties that relate to Object Node elements in activity diagrams.

ShowName

The property ShowName determines what text is opened at the top of Object Node elements. The possible values are:

- Represents - Only the type represented by the element is displayed. If no type was selected for the Represents field, the element name is displayed.
- Name_only - Both the name of the element and the type it represents are displayed.
- Label - The label of the element is displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Object Node elements). When you change the value of this property, the display of any new Object Node elements are affected, but the display of Object Node elements already on the diagram remains as is.

Default = Name_only

ShowStereotype

The ShowStereotype property determines if, and how, an stereotypes of the element are displayed in a diagram. The possible values are:

- Label - The stereotypes of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is displayed.
- None - The stereotypes of the element are not displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Call Operation elements). When you change the value of this property, the display of any new Call Operation elements are affected, but the display of Call Operation elements already on the diagram remains as is.

Default = None

OnOffSwitch

The OnOffSwitch metaclass contains properties that determine the appearance and behavior of on/off switch controls on activity diagrams.

Direction

The Direction property determines whether the on/off switch controls are used to input data, display data, or both. The possible values are:

- In - The on/off switches are only used to input data for the attribute to which it is bound.
- Out - The on/off switches are only used to display data for the attribute to which it is bound.
- InOut - The on/off switches are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for on/off switch elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the on/off switch.
- BindedElement - The name of the attribute that is bound to the on/off switch.
- Name - The name of the on/off switch element.
- None - No text is displayed.

Default = Name

Partition

The Partition metaclass contains properties that control the appearance of partitions (instance lines) in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,0,255)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 0,0,255)

name_color

The name_color property specifies the default color of names of graphical items.

(Default = 0,0,255)

PartitionFrame

The PartitionFrame metaclass contains properties that control the appearance of partition frames in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,0,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

(Default = 1)

PushButton

The PushButton metaclass contains properties that determine the appearance and behavior of push button controls on activity diagrams.

ButtonFont

The ButtonFont property lets you select the font to use for the text on the face of a push button control.

To change the value of the property, click the "..." button in the box next to the property value to open the Font window. The value of the property affects both buttons already on the activity diagram and new buttons added to the diagram. (The display of buttons already on the activity diagram changes only after

you refresh the diagram.)

Default = Arial 10 NoBold NoItalic

ShowName

The ShowName property determines whether or not a caption is displayed for push button elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the push button.
- BindedElement - The name of the attribute that is bound to the push button.
- Name - The name of the push button element.
- None - No text is displayed.

Default = Name

ReferenceActivity

The ReferenceActivity metaclass contains properties that control the appearance of references in activity diagrams.

showName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase

- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

Requirement

The Requirement metaclass contains properties that control the appearance of requirements in activity diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties dialog or directly in the .prp file. Rather, you should use the Display Options for an element to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

RequirementNotation

The RequirementNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Requirement:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles. If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available

options:

- Name
- Description
- Label - the label provided for the element
- Specification - the content of the specification field; relevant for elements such as constraints

(Default = Description)

ShowForm

Determines how note-like elements are displayed. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

The various combinations of possible values are as follows:

- Plain, Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- Statechart::Note
- ObjectModelGe::Requirement
- ObjectModelGe::Comment
- ObjectModelGe::Constraint
- UseCaseGe::Requirement
- UseCaseGe::Comment
- UseCaseGe::Constraint
- Activity_diagram::Requirement
- Activity_diagram::Comment
- Activity_diagram::Constraint
- Note, Pushpin (default is Note)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Note
- UseCaseGe::Note
- Activity_diagram:Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows

you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends
- UseCaseGe::Depends

- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase
- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

Requirement

The Requirement metaclass contains properties that control the appearance of requirements in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

(Default = 1)

name_color

The name_color property specifies the default color of names of graphical items.

(Default = 0,0,0)

SelectorConnector

The SelectorConnector metaclass contains properties that control the appearance of selector connectors in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,255)

UseFillcolor

The UseFillcolor property is a Boolean value that specifies whether to use the fill color specified for that type of connector or port.

(Default = True)

SendAction

The SendAction metaclass contains properties that relate to Send Action elements in activity diagrams.

ShowNotation

The property ShowNotation determines what text is opened on Send Action elements in an activity diagram. The possible values are:

- Name - The name of the element is displayed.
- Label - The label of the element is displayed.
- Event - The name of the event selected is displayed.
- FullNotation - In addition to the name of the event selected, Rational Rhapsody displays the name of the target selected and the argument values you provided.

Note that this property can only be set at the diagram level or higher (not at the level of individual Send Action elements). When you change the value of this property, the display of any new Send Action elements are affected, but the display of Send Action elements already on the diagram remains as is.

Default = FullNotation

ShowStereotype

The ShowStereotype property determines if, and how, an stereotypes of the element are displayed in a diagram. The possible values are:

- Label - The stereotypes of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is displayed.
- None - The stereotypes of the element are not displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual Call Operation elements). When you change the value of this property, the display of any new Call Operation elements are affected, but the display of Call Operation elements already on the diagram remains as is.

Default = None

Slider

The Slider metaclass contains properties that determine the appearance and behavior of slider controls on activity diagrams.

Direction

The Direction property determines whether slider controls are used to input data, display data, or both. The possible values are:

- In - The sliders are only used to input data for the attribute to which it is bound.

- Out - The sliders are only used to display data for the attribute to which it is bound.
- InOut - The sliders are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for slider elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the slider.
- BindedElement - The name of the attribute that is bound to the slider.
- Name - The name of the slider element.
- None - No text is displayed.

Default = Name

State

The State metaclass contains properties that control the appearance of states in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,64)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

(Default = 1)

name_color

The name_color property specifies the default color of names of graphical items.

(Default = 0,0,0)

StateDiagram

The StateDiagram metaclass contains a property that controls the appearance of state diagrams.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

(Default = 218,218,218)

SubActivityState

The SubActivityState metaclass properties control the appearance of subactivity states in activity diagrams.

line_style

The line_style property specifies the type of line used for a graphical item. The possible values are:

- straight_arrows - a straight line.
- rectilinear_arrows - rectilinear lines with right-angled corners placed at appropriate locations, depending on the start and end points of the line.
- spline_arrows - curved line without corners.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is straight_arrows:

- UseCaseGe::Inheritance
- Activity_diagram::DefaultTransition
- Statechart::Depends

For the following subject::metaclass combinations, the default value is spline_arrows:

- Statechart::DefaultTransition
- Activity_diagram::SubActivityState

showName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends

- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase
- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:

- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element are displayed in a diagram. The possible values are:

- Label - The stereotype of the element are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element are displayed.
- None - The stereotype of the element are not displayed.

The default value for this property varies for the different elements. For the following subject::metaclass combinations, the default value is None:

- Statechart::DefaultTransition
- Activity_diagram::Action
- Activity_diagram::ActionBlock
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity

For the following subject::metaclass combinations, the default value is Label:

- Statechart::Requirement
- Statechart::Depends
- ObjectModelGe::Depends
- ObjectModelGe::Inheritance
- ObjectModelGe::Actor
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::Depends
- Activity_diagram::SubActivityState

Swimlane

The Swimlane metaclass properties control the that control the appearance of swimlanes in activity diagrams.

ShowName

The property ShowName determines what text is opened at the top of a swimlane. The possible values are as follows:

- Represents - Only the class, package, or component represented by the swimlane is displayed. If the Represents field was left blank, the swimlane name is displayed.

- Name_only - Both the name of the swimlane and the class, package, or component it represents are displayed.
- Label - The label of the swimlane is displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual swimlanes).

When you change the value of this property, the display of any new swimlanes is affected, but the display of swimlanes already on the diagram remains as is.

Default = Name_only

ShowStereotype

The ShowStereotype property determines if, and how, the stereotypes of a swimlane are displayed in a diagram. The possible values are:

- Label - The stereotypes of the swimlane are displayed as a text label.
- Bitmap - The bitmap image associated with the stereotype of the swimlane is displayed.
- None - The stereotypes of the swimlane are not displayed.

Note that this property can only be set at the diagram level or higher (not at the level of individual swimlanes).

When you change the value of this property, the display of any new swimlanes is affected, but the display of swimlanes already on the diagram remains as is.

Default = Label

TerminationConnector

The TerminationConnector metaclass contains properties that control the appearance of termination connectors in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 0,128,0)

UseFillcolor

The UseFillcolor property is a Boolean value that specifies whether to use the fill color specified for that type of connector or port.

(Default = True)

TextBox

The TextBox metaclass contains properties that determine the appearance and behavior of text box controls on activity diagrams.

Direction

The Direction property determines whether text box controls are used to input data, display data, or both. The possible values are:

- In - The text boxes are only used to input data for the attribute to which it is bound.
- Out - The text boxes are only used to display data for the attribute to which it is bound.
- InOut - The text boxes are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for text box elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the text box.
- BindedElement - The name of the attribute that is bound to the text box.
- Name - The name of the text box element.
- None - No text is displayed.

Default = Name

Transition

The Transition metaclass contains a property that controls the appearance of transition lines in activity diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 255,0,0)

label_color

The label_color property is currently unused.

LoopTransition

Statechart::Transition,0,127,255

(Default = 0,127,255)

line_style

The line_style property specifies the default line style for a graphical item.

The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

(Default = spline_arrows)

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class are displayed with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

The various combinations of possible values are as follows:

- Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- Statechart::State
- Name, Label, None (default is Name)
- Used for following subject::metaclass combinations:
- Statechart::Transition
- Statechart::DefaultTransition
- ObjectModelGe::Aggregation
- ObjectModelGe::Composition
- ObjectModelGe::Association
- ObjectModelGe::Link
- UseCaseGe::Association
- Activity_diagram::Transition
- Activity_diagram::DefaultTransition
- Activity_diagram::ReferenceActivity
- Activity_diagram::SubActivityState
- Activity_diagram::ActivityParameter
- Name, Label, None (default is None)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Inheritance
- ObjectModelGe::Depends
- UseCaseGe::Depends
- UseCaseGe::Inheritance
- Activity_diagram::ActivityParameter
- Activity_diagram::Depends
- Statechart::Depends
- Full_path, Relative, Name_only, Label (default is Relative)
- Used for following subject::metaclass combinations:
- Statechart::Requirement
- ObjectModelGe::Class
- ObjectModelGe::Object
- ObjectModelGe::UseCase
- ObjectModelGe::Actor
- ObjectModelGe::Requirement
- UseCaseGe::Actor
- UseCaseGe::Association
- UseCaseGe::Requirement
- UseCaseGe::UseCase

- Activity_diagram::Requirement
- Name, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Comment
- ObjectModelGe::Comment
- UseCaseGe::Comment
- Activity_diagram::Comment
- Name, Specification, Description, Label (default is Description)
- Used for following subject::metaclass combinations:
- Statechart::Constraint
- ObjectModelGe::Constraint
- UseCaseGe::Constraint
- Activity_diagram::Constraint
- Full_path, Relative, Name_only, Label (default is Name_only)
- Used for following subject::metaclass combinations:
- ObjectModelGe::Package
- ObjectModelGe::PrimitiveOperation
- ObjectModelGe::Attribute
- ObjectModelGe::Type
- Name, Label (default is Name)
- Used for following subject::metaclass combinations:
- Activity_diagram::Swimlane

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams.

The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

(Default = None)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

(Default = 1)

Ada_CG

The Ada_CG subject contains several metaclasses for operating system environments and the following general metaclasses:

- Component
- Argument
- Attribute
- Class
- Dependency
- File
- Framework
- GNAT
- GNATVxWorks
- INTEGRITY
- INTEGRITY5
- Multi4Win32
- MultiWin32
- OBJECTADA
- Operation
- Package
- Port
- RAVEN_PPC
- Relation
- SPARK
- Type

Argument

The Argument metaclass contains properties that control how arguments are generated in code.

AccessTypeUsage

This property defines the access type.

Default = None

AsAccess

The AsAccess property sets the mode of a parameter to be access (as opposed to in, out, or in out). Note that access parameters are supported by Ada95, not Ada83.

Default = Cleared

ClassWide

The ClassWide property determines whether a class-wide modifier is generated for the argument.

Default = Cleared

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

Attribute

The Attribute metaclass contains properties that control attributes of code generation, such as whether to generate accessor operations.

Accessor

The Accessor property is ignored by Rational Rhapsody.

Default = Get_$attribute:c

AccessorGenerate

The AccessorGenerate property specifies whether to generate accessor operations for attributes. The possible values are as follows:

- Checked - A get() method is generated for the attribute.
- Cleared - A get() method is not generated for the attribute.

Setting this property to Cleared is one way to optimize your code for size.

Default = Cleared

AccessTypeUsage

This property defines the access type.

Default = None

AccessorVisibility

The AccessorVisibility property specifies the access level of the generated accessor for attributes. This enables you to define the access level of an accessor operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute’s access level for the accessor.
- public - Set the accessor’s access level to public.
- private - Set the accessor’s access level to private.
- protected - Set the accessor’s access level to protected.
- default - Set the accessor’s access level to default.

Default = fromAttribute

AttributeInitializationFile

The AttributeInitializationFile property specifies how static const attributes are initialized. In Rational Rhapsody, you can initialize these attributes in the specification file or directly in the initialization file. This property is analogous to the VariableInitializationFile property for global const variables. The possible values are as follows:

- Default - The attribute is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize constant attributes in the implementation file.
- Specification - Initialize constant attributes in the specification file.

Default = Default

ConstantVariableAsDefine

This property is a Boolean value that determines whether the variable, defined as constant in file or package, is generated using a #define macro. Otherwise, it is generated using the const qualifier.

Default = Cleared

DeclarationPosition

The DeclarationPosition property enables you to control the declaration order of attributes. The possible values are as follows:

- Default - Similar to the AfterClassRecord setting, with the following difference:
- For static attributes defined in a class with the property ADA_CG::Attribute::Visibility set to Public, these attributes are generated after types whose ADA_CG::Type::Visibility property is set to Public.
- You should not use this setting for new models. For more information, see the Sodius documentation for Ada.
- BeforeClassRecord - Generate the attribute immediately before the class record.
- AfterClassRecord - Generate the attribute immediately after the class record.
- StartOfDeclaration - Generate the attribute immediately after the start of the section (private or public part of the specification, or package body).
- EndOfDeclaration - Generate the attribute immediately before the end of the section (private or public part of the specification, or package body).

Default = Default

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered $Arguments - The operation argument’s description operations, constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef_DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

InitializationStyle

The InitializationStyle property specifies the initialization style used for attributes. When you specify an initial value for an attribute, Rational Rhapsody initializes the attribute based on the value of this property. In Rational Rhapsody Developer for C++, the possible values are as follows:

- ByInitializer - Initialize the attribute in the initializer (a(y)). This is the default value.
- If the initialization style is ByInitializer, the attribute initialization should be done after the user initializer, in the same order as the order of attributes in the code.
- ByAssignment - Initialize the attribute in the constructor body (a = y).

In Rational Rhapsody Developer for Java, the possible values are as follows:

- InClass - Initialize the attribute in the class declaration. This is the default value.
- InConstructor - Initialize the attribute in each of the class constructors.

In Rational Rhapsody Developer for C, the attribute is initialized in the initializer body.

Default = ByInitializer

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations

- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for Ada If the operation is marked as inline, the package specification includes an inline pragma after the declaration. For example: pragma inline (operation name); The two possible values for Rational Rhapsody Developer for Ada are as follows:

- none
- use_pragma

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased.

Default = Cleared

IsMutable

The boolean property IsMutable allows you to specify that an attribute is a mutable attribute.

Default = False

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.

- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

Mutator

The Mutator property is ignored by Rational Rhapsody.

Default = Set_$attribute:c

MutatorGenerate

The MutatorGenerate property specifies whether to generate mutators for attributes. The possible values are as follows:

- Smart - Mutators are not generated for attributes that have the Constant modifier.
- Always - Mutators are generated, regardless of the modifier.
- Never - Mutators are not generated.

Default = Never

MutatorVisibility

The MutatorVisibility property specifies the access level of the generated mutator for attributes. This enables you to define the access level of a mutator operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute's access level for the mutator.
- public - Set the mutator's access level to public.
- private - Set the mutator's access level to private.
- protected - Set the mutator's access level to protected.
- default - Set the mutator's access level to default.

Default = fromAttribute

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code.

Default = ""*

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG,

set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

Using SpecificationProlog to Inherit from an External Object You can use the SpecificationProlog property to inherit from external classes, including template classes. For example, you can define an external template class named DataStore in Rational Rhapsody as a placeholder for the external code using the UseAsExternal property. You can then use the DataStore object in an object model diagram. If a subclass inherits from DataStore, you can set the SpecificationProlog property for the subclass to: class D : public DataStoreD // The cr// actually eliminates the generated definition for the subclass, so its generated code looks like this: class D : public DataStoreD // class D

VariableInitializationFile

The VariableInitializationFile property specifies how global constant variables are initialized. You can initialize these variables in the specification file. You can use these variables as compile-time constants that can be used to define array sizes, for example. Rational Rhapsody automatically identifies constant variables with const. By modifying this property, you can choose the initialization file directly. The possible values are as follows:

- Default - The variable is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize global constant variables in the implementation file.
- Specification - Initialize global constant variables in the specification file.

Default = Default

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language.

Default = Public

Class

The Class metaclass contains properties that affect the generated classes.

AccessTypeName

The AccessTypeName property specifies the name of the access type generated for the class record.

Default = Empty string

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active classes. The possible values are as follows:

- A string - Specifies the message queue size for an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

Default = Empty string

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects. The possible values are as follows:

- Any integer - Specifies that a stack of that size is allocated for active objects.
- An empty string (blank) - If not specified, the stack size is set in an operating system-specific manner, based on the value of the ActiveStackSize property for the framework.

Default = Empty string

ActiveThreadName

The ActiveThreadName property specifies the name of the active thread. This facilitates debugging in complex environments in which many threads are constantly being created and deleted on-the-fly. This property is effective only in the pSoSystem (both PPC and X86) and VxWorks environments. In pSoSystem, the thread name is truncated to three characters. The animation thread name is not taken from the active thread name. The possible values are as follows:

- A string - Names the active thread.
- An empty string (blank) - The value is set in an operating system-specific manner, based on the value of the ActiveThreadName property for the framework.

Default = Empty string

ActiveThreadPriority

The ActiveThreadPriority property specifies the priority of active class threads. The possible values are as follows:

- A string - Specifies thread priority of an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

Default = Empty string

AdditionalBaseClasses

The AdditionalBaseClasses property enables you to add inheritance from external classes to the model.

Default = Empty string

AdditionalNumberOfInstances

The AdditionalNumberOfInstances property is a string that specifies the size of the local heap allocated for events when the current pool is full. Triggered operations use the event properties. This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during runtime. All events are dynamically allocated during initialization. Once allocated, a thread's event queue remains static in size. The possible values are as follows:

- An empty string (blank) - No additional memory is to be allocated when the initial memory pool is exhausted.
- n (a positive integer) - Specifies the size of the array allocated for additional instances.

Default = Empty string

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to Cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to Cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to Cleared, all the arguments are not animated.
- If the AnimateArguments property is set to Cleared, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = Checked

BaseNumberOfInstances

The BaseNumberOfInstances property is a string that specifies the size of the local heap memory pool allocated for either:

- Instances of the class (ADA_CG::Class)

- Instances of the event (ADA_CG::Event)
- This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during run time. All instances of events are dynamically allocated during initialization. Once allocated, a thread's event queue remains static in size.

Triggered operations use the properties defined for events. When the memory pool is exhausted, an additional amount, specified by the AdditionalNumberOfInstances property, is allocated. Memory pools for classes can be used only with the Flat statechart implementation scheme. The possible values are as follows:

- An empty string (blank) - Memory is always dynamically allocated.
- n (positive integer) - An array is allocated in this size for instances.

The related properties are as follows:

- AdditionalNumberOfInstances - Specifies the number of instances to allocate if the pool runs out.
- ProtectStaticMemoryPool - Specifies whether the pool should be protected (to support a multithreaded environment)
- EmptyMemoryPoolCallback - Specifies a user callback function to be called when the pool is empty. This property should be used instead of the AdditionalNumberOfInstance property for error handling.
- EmptyMemoryPoolMessage - When set to true, this property causes a message to be displayed if the pool runs out of memory in instrumented mode.

Default = Empty string

ComplexityForInlining

The ComplexityForInlining property specifies the upper bound for the number of lines in user code that are allowed to be inlined. User code is the action part of transitions in statecharts. For example, using the value 3, all transitions with actions consisting of three lines or fewer of code are automatically inlined in the calling function. Inlining is replacing a function call in the generated code with the actual code statements that make up the body of the function. This optimizes the code execution at the expense of an increase in code size. For example, increasing the number of lines that can be inlined from 3 to 5 has shortened the code execution time in some cases up to 10%. This property applies only to the Flat implementation scheme for statecharts.

Default = 0

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to "MYDLL_API." The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class.

Default = Empty string

DeclarationPosition

This property allows you to control the declaration order of attributes.

The possible values are as follows: Default - Similar to the AfterClassRecord setting, with the following difference: For static attributes defined in a class with the property ADA_CG::Attribute::Visibility set to Public, these attributes are generated after types whose ADA_CG::Type::Visibility property is set to Public. As a general rule, you should not use this setting for new models.

- BeforeClassRecord - Generate the attribute immediately before the class record.
- AfterClassRecord - Generate the attribute immediately after the class record.
- StartOfDeclaration - Generate the attribute immediately after the start of the section (private or public part of the specification, or package body).
- EndOfDeclaration - Generate the attribute immediately before the end of the section (private or public part of the specification, or package body).

Default = EndOfDeclaration

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified the element tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the

ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

Destructor

The Destructor property controls the generation of virtual destructors in C++. The property exists for C for historical reasons, with a single value of auto, but it has no effect on the generated C code. The possible values are as follows:

- auto - A virtual destructor is generated for an object only if it has at least one virtual function.
- virtual - A virtual destructor is generated in all cases.
- abstract - A virtual destructor is generated as a pure virtual function.
- common - A nonvirtual destructor is generated.

Default = auto

Embeddable

The Embeddable property is a Boolean property that specifies whether a class can be allocated by value (nested) inside another class or package. For example, if the Embeddable property is True, 20 instances of a class A can be allocated inside another class using the following syntax: A itsA[20]; The possible values are as follows:

- True - The object can be allocated by value inside a composite object or package. The object declaration and definition are generated in the specification file of the composite.
- False - The object cannot be embedded inside another object. The object declaration and definition are generated in the implementation file of the composite.

The Embeddable property is used with the EmbeddedScalar and EmbeddedFixed properties to determine how to generate code for an embedded object. The Embeddable property must be set to True for either of those properties to take effect. It is also closely related to the ImplementWithStaticArray property, which also needs to be set in order to support by-value allocation. To generate C-like code in C++, set the Embeddable property to True. Relations can be generated by value only under the following circumstances:

- The Embeddable property of the nested class is set to True.
- The multiplicity of the relation is well-defined (not “*”).
- The ImplementWithStaticArray property of the component relation is set to FixedAndBounded.

When the Embeddable property is False:

- The attributes of the object are encapsulated. Clients of the object are forced to use it only via its operations, because there is no direct access to its attributes.
- Dynamic allocation must be used. The compiler does not know how to statically allocate an object when its declaration is not visible.
- The nested object cannot be reactive. This is because of the reactive macros. There is a complex workaround for this issue.

Default = True

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- True - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for the object or object type.
- False - Events are dynamically allocated during initialization, but not during run time. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during run time.

If you are managing your own memory pools, set this property to False and call CPPReactive_gen() directly. The following example shows how to call RiCReactive_gen() directly to send a static event to a reactive object A, when using a member function of A genStaticEv2A(): void A_genStaticEv2A(struct A_t* const me) { { /*#[operation genStaticEv2A() */ static struct ev _ev; ev_Init(_ev); RiCEvent_setDeleteAfterConsume(((RiCEvent*)_ev), RiCFALSE); (void) RiCReactive_gen(me-ric_reactive, ((RiCEvent*)_ev), RiCFALSE); /*#]*/ } }

Alternatively, you can use internal memory pools by setting the property BaseNumberOfInstances, which results in the use of framework memory pools. If you use the framework memory pools, do not disable the Create() and Destroy() methods because these methods are used to manage the memory pool. When you disable the generation of the Create() and Destroy() methods, you can still inject events in animation by supplying an alternate function to get an event instance. To do this, set the AnimInstanceCreate property.

Default = True

EnableUseFromCPP

The EnableUseFromCPP property specifies whether to wrap C operations with an appropriate extern C {} wrapper to prevent problems when code is compiled with a C++ compiler. Wrapping C code with extern C enables you to include C code in a C++ application. Note that the structure definition for the object is not wrapped - only the functions are. For example, if the EnableUseFromCPP is set to True for an object, the following wrapper code is generated for its operations:

#ifdef __cplusplus extern "C" { #endif /* __cplusplus */ /* Operations */ #ifdef __cplusplus } #endif /* __cplusplus */

Default = False

Final

The Final property, when set to Cleared, specifies that the generated record for the class is a tagged record. This property applies to ADA95. Default = Cleared

GenerateAccessType

The GenerateAccessType property determines which access types are generated for the class. The possible values are as follows:

- None - Access types are not generated.
- Standard - An access type is generated.
- General - General access types are generated.

Default = None

GenerateDestructor

The GenerateDestructor property specifies whether to generate a destructor for a class.

Default = True

GenerateRecordType

This property determines whether the class record is generated.

Default = Checked

HasUnknownDiscriminant

This property determines whether an unknown discriminant (encased in quotation marks) is generated for this class.

Default = Cleared

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces.

Default = Empty string>

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.

- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body.

Default = Empty MultiLine

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body.

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

In

The In property specifies how code is generated when the type is used with an argument that has the modifier "In." When used with the "In" modifier, classes are mapped as .

InitCleanUpRelations

The InitCleanUpRelations property specifies whether to generate initRelations() and cleanUpRelations() operations for sets of related global instances. This property applies only to composites and global relations.

Default = True

InitializationCode

The InitializationCode property adds the specified initialization code in the body of the class. A non-abstract class can have initialization code that is executed during elaboration of the associated package.

Default = Empty MultiLine

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier "InOut."

InstanceDeclaration

The InstanceDeclaration property specifies how instances are declared in code. The default value for C is as follows: struct $cname$suffix In the generated code, the variable $cname is replaced with the object (or object type) name. The variable $suffix is replaced with the type suffix “_t,” if the object is of implicit type.

IsCompletedOperation

The IsCompletedOperation specifies whether state_IS_COMPLETED operations are generated as functions or macros (using #define). The possible values are as follows:

- Plain - state_IS_COMPLETED operations are generated as functions (pre-V4.2 behavior). This is the default value.
- Inline - state_IS_COMPLETED operations are generated using #define macros, if the body contains only a return statement.

Default = Plain

IsInOperation

The IsInOperation specifies how state_IN methods are generated.

IsLimited

The IsLimited property determines whether the class or record type is generated as limited.

Default = Cleared

IsNested

The IsNested property specifies whether to generate the class or package as nested.

Default = Cleared

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private.

Default = Cleared

IsReactiveInterface

The IsReactiveInterface property modifies the way reactive classes are generated. It has the following effects:

- Virtual inheritance from OMReactive
- Prevents instrumentation
- Prevents the thread argument and the initialization code (setting the active context) in the class constructor
- Creates a pure-virtual destructor (by default)

This property affects only classes that declare themselves as interfaces by having a stereotype with a name that contains the word “interface” (case-insensitive). In previous versions of Rational Rhapsody, a class could inherit from a single reactive class only, regardless of whether it was an interface or implementation class. Beginning with Version 4.0.1 MR2, a class can inherit (implement) several reactive interfaces. In Rational Rhapsody Developer for C++, you must explicitly designate reactive interfaces because the code generator applies special translation rules involving multiple inheritance from the Rational Rhapsody framework. You can designate a reactive interface in two ways:

- Set the property ADA_CG::Class::IsReactiveInterface to true.
- Use the predefined stereotype Reactive_interface. This stereotype uses stereotype-based code generation in order to automatically apply the correct property value.

Alternatively, you can define another stereotype (such as PortSpec;) that sets IsReactiveInterface to true and use that stereotype. A class is considered reactive if it meets all the following conditions:

- The ADA_CG::Framework::ReactiveBase property is not empty.
- The ADA_CG::Framework::ReactiveBaseUsage property is set to true.
- One or more of the following conditions are true:

- The class has a statechart or activity diagram.
- The class is a composite class.
- The class has event receptions or triggered operations.

Default = True

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

MaximumPendingEvents

The MaximumPendingEvents property specifies the maximum number of events that can be simultaneously pending in the event queue of the active class. The possible values are as follows:

- –1 - Memory is dynamically allocated.
- Positive integer - Specifies the maximum number of events.

Default = –1

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or

package.

Default = Public

ObjectTypeAsSingleton

The ObjectTypeAsSingleton property enables you to generate singleton code for object-types and actors. This functionality enables you to save a singleton-type (actor) in its own repository unit, and manage that unit using a configuration management tool. Set this property for a single object-type or higher. An object-type is generated as a singleton when all of the following conditions are met:

- The object-type has the «Singleton» stereotype.
- There is one and only one object of the object-type and the object multiplicity is 1.
- The ObjectTypeAsSingleton property is set to True.

Note that when you expose a singleton object (for example, by creating a singleton object-type), Rational Rhapsody also modifies the code generated for the singleton.

Default = False

OptimizeStatechartsWithoutEventsMemoryAllocation

The OptimizeStatechartsWithoutEventsMemoryAllocation property determines whether the generated code uses dynamic memory allocation for statecharts that use only triggered operations.

Default = Cleared

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out."

PrivateInherits

The PrivateInherits property, when set for a particular class, contains the names of the base classes from which the class privately inherits. For example, if a class B is a subclass of a class A and PrivateInherits is set to A for class B, the code generated for B contains a declaration showing inheritance from A, as follows: class B: private A { ... }; This property is interpreted as a string list. To combine classes for multiple inheritance by concatenation, separate the class names using commas. For example, if you want to have class C inherit privately from classes A and B, set the PrivateInherits property for class C to “A,B.”

Default = Empty string

ReactiveThreadSettingPolicy

The ReactiveThreadSettingPolicy property enables you to specify how threads are set for reactive classes. The possible values are as follows:

- Default - During code generation, Rational Rhapsody adds a thread argument to the constructor.
- MainThread - Rational Rhapsody does not add an argument; the thread is set to the main thread.
- UserDefined - Rational Rhapsody does not add an argument; you must set the value for the thread yourself.

Default = Default

RecordTypeName

The RecordTypeName property specifies the name of the class record type. If this is not set, Rational Rhapsody uses class_name>_t.

Default = Empty string

RelativeEventDataRecordTypeComponentsNaming

This property enables relative naming of event data record type components that represent events and triggered operation parameters. If this is Checked, no events or triggered operations will share argument names because they would generate record components with the same name (which would not compile).

Default = Cleared

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type.

SingletonExposeThis

The SingletonExposeThis property, when set to Cleared, specifies that all non-static methods are considered as static methods and will not have a this parameter passed in.

Default = Cleared

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification.

Default = Empty MultiLine

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification.

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef_DEBUG cr.
- Set SpecificationEpilog to #endif.

- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces.

Default = Empty string

TaskBody

The TaskBody property enables you to define an alternate task body for ADA Task and ADA Task Type classes.

Default = Empty string

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

- In
- InOut
- Out

*Default = $type**

VirtualInherits

The VirtualInherits property, when set for a particular class, contains the names of base classes from which the inherits from as virtual. For example, if class B is a subclass of class A and VirtualInherits is set to “A” for class B, the code generated for B contains a declaration showing inheritance from A, as follows:

class B: virtual public A { ... };

This property is interpreted as a string list. To combine classes for multiple inheritance by concatenation. separate the names with commas. For example, if you want to have class C inherit virtually from classes A and B, set the VirtualInherits property for class C to “A,B.”

Default = Empty string

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

Default = Public

Component

The Component metaclass contains properties that affect the Ada component.

AdaVersion

This property indicates what version of ADA is being used.

Default = Ada95

ClassStateDeclaration

The ClassStateDeclaration property supports C compilers that cannot handle enum declarations inside struct declaration. The possible values are as follows:

- InClassDeclaration - Generate the reactive statechart enum declaration in the class declaration.
- BeforeClassDeclaration - Generate the reactive class statechart enum declaration before the declaration of the class.

Default = InClassDeclaration

Configuration

The Configuration metaclass contains properties that affect the configuration.

ClassStateDeclaration

The ClassStateDeclaration property supports C compilers that cannot handle enum declarations inside struct declaration. The possible values are as follows:

- InClassDeclaration - Generate the reactive statechart enum declaration in the class declaration.
- BeforeClassDeclaration - Generate the reactive class statechart enum declaration before the declaration of the class.

Default = InClassDeclaration

CodeGeneratorTool

The CodeGeneratorTool property specifies which code generation tool to use for the given configuration. The possible values are as follows:

- External - Use the registered, external code generator.
- Internal - Use the Rational Rhapsody internal code generator.

Default = External.

ContainerSet

The ContainerSet property specifies the container set used to implement relations.

Default = None

DefaultActiveGeneration

The DefaultActiveGeneration property specifies whether the default active class is created, as well as the classes for which it acts as the active context. The possible values are as follows:

- Disable - The default active singleton is not created.
- ReactiveWithoutContext - The default active singleton is created if there are reactive classes that consume events and do not have an active context explicitly specified. The default active singleton can handle only these classes.
- All - The default active singleton is generated if there is at least one event-consuming reactive class and the active singleton can handle all reactive classes that consume events - even those reactive classes that specify another active class as their active context.

Default = ReactiveWithoutContext

DefaultImplementationDirectory

The DefaultImplementationDirectory property specifies the relative path to the default directory for generated implementation files. The value of this property is added after the configuration path. Consider

the following case:

- File C.cpp is an implementation of class C mapped to a folder Foo.
- The active configuration (cfg) is under component cmp.
- DefaultImplementationDirectory is set to “src”

Rational Rhapsody generates C.cpp to root>\cmp\cfg\src\Foo. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

Default = Empty string

DefaultSpecificationDirectory

The DefaultSpecificationDirectory property specifies the relative path to the default directory for generated specification files. The value of this property is added after the configuration path. Consider the following case:

- File B.h is a specification of class B that is not mapped to any file.
- The active configuration (cfg) is under component cmp.
- DefaultSpecificationDirectory is set to “inc”

Rational Rhapsody generates B.h to root>\cmp\cfg\inc. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

Default = Empty string

DependencyRuleScheme

The DependencyRuleScheme property specifies how dependency rules should be generated in the makefile. The possible values are as follows:

- Basic - Generates only the local implementation and specification files in the dependency rule in the makefile.
- ByScope - In addition to generating the same files as the Basic option, generates the specification files of related elements (dependencies, associations, generalizations, and so on) that are in the scope of the active component.
- Extended - In addition to generating the same files as the ByScope option, generates the specification files of related external elements (specified using the properties CG::Class/Package::UseAsExternal) and elements that are not in the scope of the active component.

Default = ByScope

DescriptionBeginLine

This property enables you to specify the prefix for the beginning of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation. This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected. The following table lists the default value for each language.

Language Edition Default Value C "//" C++ ""

When you set this property, you should check the value of the C_CG::DiffDelimiter property - if the same prefix is used, Rational Rhapsody will not update the generated code when the description is modified. If both DescriptionBeginLine and DiffDelimiter use the same prefix, modify the values of the following properties under C_CG::File:

DiffDelimiter ImplementationHeader SpecificationHeader

DescriptionEndLine

This property enables you to specify the prefix for the end of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation. This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected. The following table lists the default value for each language.

Language Edition Default Value C "/*" C++ "*/"

EmptyArgumentListName

The EmptyArgumentListName specifies the string generated for the argument list when an operation has no arguments. For example, if you set this value to “void”, for an operation foo that has no arguments, Rational Rhapsody generates the following code:

int foo (void){...}

Default = Empty string

Environment

The Environment property determines the target environment for a configuration. Generated code is targeted for that environment. See the Release Notes for the environments supported by Rational Rhapsody "out-of-the-box." "Out-of-the-box" support means that Rational Rhapsody includes a set of preconfigured code generation properties for the environment and precompiled versions of the relevant OXF libraries. The precompiled OXF libraries have been fully tested. You can also add new environments, for example if you want to generate code for another RTOS. This involves retargeting the OS wrapper files in the Rational Rhapsody framework and creating a new set of code generation properties for the target environment.

Default = GNAT

ExternalGenerationTimeout

The ExternalGenerationTimeout property specifies how long, in seconds, Rational Rhapsody waits for the each class in the configuration scope to complete so you can once again make changes to the model. This property applies to both the full-featured external generator and makefile generator. For example, if you set this property to 2 and you have 10 classes, Rational Rhapsody sets a timeout of 20. If the external code generator does not complete generation in this timeframe, Rational Rhapsody displays a message in the output window saying that the generator is not responding, and you are allowed to make changes to the model. If you set this property to 0, Rational Rhapsody will not time out the generation session, and waits for the code generator to complete its task - even if it takes forever. Rational Rhapsody waits for a notification from the full-featured external code generator, or for the process termination of a makefile generator.

Default = 0

ExternalGeneratorFileMappingRules

The ExternalGeneratorFileMappingRules property specifies whether the external code generator uses the same file mapping and naming scheme (mapping rules) as Rational Rhapsody. If the mapping rules are different , the external generator must implement handlers to the GetFileName, GetMainFileName, and GetMakefileName events that Rational Rhapsody runs to get a requested file name and path. The possible values are as follows:

- AsRhapsody - The external generator uses the same mapping rules as Rational Rhapsody.
- DefinedByGenerator - The external generator has its own mapping rules.

Default = DefinedByGenerator

GenerateAnnotationsForNonSPARKConfigurations

The GenerateAnnotationsForNonSPARKConfigurations property specifies whether or not to generate annotations for this configuration.

Default = Cleared

GenerateDirectoryPerModelComponent

The GenerateDirectoryPerModelComponent property specifies whether to generate a separate directory for each package in the component. The possible values are as follows:

- True - Rational Rhapsody creates a separate directory for each package in the component.
- False - A separate directory is not created for each package.

Default = Checked

GeneratorExtraPropertyFiles

The GeneratorExtraPropertyFiles property launches the default Text Editor allowing the user to edit the $OMROOT\CodeGenerator\GenerationRules\LangC\RiC_CG.ini file.

Default = $OMROOT\..\Sodius\RiA_CG\RiA_CG.ini

GeneratorRulesSet

The GeneratorRulesSet property enables you to specify your own rules set.

Default = $OMROOT\..\Sodius\RiA_CG\compiled_rules\MDWGen.jar

GeneratorScenarioName

The GeneratorScenarioName property specifies the scenario name for the rule, if you write your own set of code generation rules.

Default = Empty string

GenericEventHandling

The GenericEventHandling property is a Boolean value that determines whether to generate generic event-handling code. This property supports large-scale collaboration, where you might not be aware of which classes consume a base event of your part in the event hierarchy, and might not have access to parts of the model that use base events. Beginning with Rational Rhapsody 4.0, the framework base event class includes a new, virtual method that checks the event ID against the specified ID, thereby supplying a generic mechanism for events without super events. The language-specific methods are as follows: C

#define RiCEvent_isTypeOf(event, id) ((event)- == (id)) C++ virtual OMBoolean isTypeOf(short id) const {return lId ==id;}

Each generated event that has a super event will override the method to check the ID against its own ID, then calls its base event directly to continue the check. An event without a base event will return False if the ID does not equal its own. When you set the GenericEventHandling property to False, event consumption code is generated as in version 3.0.1. Setting this property affects only the way events are consumed - the override on the isTypeOf() method is still generated, to allow handling of events in components that use the generic event handling. To support complete generic event handling, you should regenerate the code for all events and reactive classes. The default value for C is False; the default value for C++ and Java is True.

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled

with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG ;cr
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG ;cr
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

InitializeEmbeddableObjectsByValue

The InitializeEmbeddableObjectsByValue property specifies whether embeddable classes and object types selected in the configuration initial instances list should be allocated by value in the main() routine.

Default = False

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation.

Default = Empty MultiLine

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters \n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

SourceListFile

The SourceListFile property specifies the name of the file containing a list of .java source files to be compiled with javac. The batch file used by the Build command (jdkmake.bat) can use the following call, rather than including a long list of source files: javac –g @files.lst This same command is generated from the following line in the MakeFileContent property for Java: javac –g @$SourceListFile If the SourceListFile property is empty, $SourceListFile is replaced with a string containing all source file names, separated by spaces (for example, "A.java B.java"). This means that if the MakeFileContent default value is not changed, you will get: javac –g @A.java B.java … If you do not want to use the file containing the list of sources, you must also change the MakeFileContent property to replace "javac –g @$SourceListFile" with "javac –g $SourceListFile".

Default = files.lst

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is

available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

Dependency

The Dependency metaclass controls the dependency for a package that defines a namespace.

AccessTypeUsage

This property defines the access type.

Default = None

CreateUseStatement

This property determines whether a use statement is added to the code after the with statement. The supplier of the dependency must be a class or type.

Default = None

GeneratePragmaElaborate

This property determines whether to generate an elaborate pragma for the supplier class in the client class or package.

Default = Cleared

GenerateOriginComment

When set to Checked, generates a comment before #include statements indicating which element "caused" the #include.

GeneratePragmaElaborateAll

This property determines whether to generate a pre-elaborate pragma for the supplier class in the client class or package.

Default = Cleared

GenerateWithClause

The GenerateWithClause property determines whether with clauses are generated for "Usage" dependencies. For example, you can generate a with clause for a package, P1, in the specification of another package, P2, using a dependency, D1, and generate a use clause for P1 in the body of P2. In addition, this functionality is useful for modeling inherited annotations across classes and packages.

Default = Checked

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

IncludeStyle

The IncludeStyle property controls the style of #include statements. Using this property, you can control the style of a specific dependency, or the entire configuration/component/project. To set the style for include files that are synthesized based on associations between model elements (for example, setting the type of some attribute to a class), add a «Usage» dependency between the elements and set this property to the appropriate value. The possible values are as follows:

Default - Use angle brackets for include statements for external elements, and quotes for include statements for other elements.

Quotes - Enclose include files in quotation marks. For example: #include "A.h"

When a compiler encounters an include file in quotes, it searches for the file in both the current directory and the directories specified in the include path. Note that the specific algorithm used is compiler-dependent.

AngledBrackets - Enclose include files in angle brackets. For example: #include A.h

When a compiler encounters an include file in angle brackets, it searches for the file only in the directories specified in the include path. If you set the property to AngledBrackets at the configuration level, you must also change the CG::File::IncludeScheme property to RelativeToConfiguration to ensure successful compilation.

Default = Default

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element’s annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef_DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

UseNameSpace

The UseNameSpace property enables you to model namespace usage. When you set a dependency to a package that defines a namespace and set this property to True, Rational Rhapsody generates a “using namespace” statement to the package namespace.

Default = False

Event

The Event metaclass contains properties that control events.

AnimInstanceCreate

The AnimInstanceCreate property affects event creation. If you set the

C_CG::Event::NoDynamicAllocAnimCreate property to False, Rational Rhapsody does not generate the event creation method, effectively disabling the ability to inject the event in animation. To enable the injection of the event, you can specify a different method to obtain an instance of the event by setting this property to the name of the method to use.

Default = Empty string

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to “MYDLL_API.” The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class.

Default = Empty string

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

Tag - The value of the specified the element tag Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)

- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- True - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for the object or object type.
- False - Events are dynamically allocated during initialization, but not during runtime. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during runtime.

Default = True

File

The File metaclass contains properties that control the generated code files.

DiffDelimiter

The DiffDelimiter property defines a symbol that is used to avoid overwriting an unchanged line of code during code generation. Use this property to avoid touching the source code file when the “diff-delimited” line has not changed. In general, fewer source files need to be recompiled if fewer source files are touched. For example, the DiffDelimiter symbol “//!” is used in the ADA_CG::File::Header property. This symbol is at the beginning of a line of code that includes the current code generation date. The code generator compares the code it would normally generate for that line (the current code generation date) to that previously generated (the last code generation date). If the date has not changed, the line is not overwritten, possibly preventing the file’s modification time from changing (being "touched").

Default = --!

Footer

The Footer property specifies a multiline footer that is added to the end of generated Java files. The default footer template is as follows:

"/********** File Path: $FullCodeGeneratedFileName **********/"

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.

- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property ADA_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

Header

The Header property specifies a multiline header that is added to the top of all generated Java files. The default header template is as follows:

/********** Rhapsody : $RhapsodyVersion Login : $Login Component : $ComponentName Configuration : $ConfigurationName Model Element : $FullModelElementName //! Generated Date : $CodeGeneratedDate File Path : $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.

- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property ADA_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationFooter

The ImplementationFooter property specifies the multiline footer to be generated at the end of implementation files.

Default = Empty MultiLine

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property ADA_CG::File::DiffDelimiter. The default DiffDelimiter value is "//!". The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

ImplementationHeader

The ImplementationHeader property specifies the multiline header that is generated at the beginning of implementation files.

Default = Empty MultiLine

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property ADA_CG::File::DiffDelimiter. The default DiffDelimiter value is "//!". The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to

the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationFooter

The SpecificationFooter property specifies the multiline footer to be generated at the end of specification files.

Default =Empty MultiLine

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property ADA_CG::File::DiffDelimiter. The default DiffDelimiter value is "//!". The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

SpecificationHeader

The SpecificationHeader property specifies the multiline header to be generated at the beginning of specification files.

Default = Empty MultiLine

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is

the name of the first element.

- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property ADA_CG::File::DiffDelimiter. The default DiffDelimiter value is "//!". The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

Framework

The Framework metaclass contains properties that affect the Rational Rhapsody framework.

ActivateFrameworkDefaultEventLoop

The ActivateFrameworkDefaultEventLoop property specifies the framework call that initializes the framework main event loop.

Default = OXF::start($Fork); The value of $Fork is calculated from the property CG::Configuration::StartFrameworkInMainThread for regular applications and from the property CORBA::Configuration::StartFrameworkInMainThread for CORBA servers. This property can be set at the configuration level or higher.

ActiveBase

The ActiveBase property specifies the superclass from which to specialize all threads, if the ActiveBaseUsage property is set to Checked.

Default = Empty string

ActiveBaseUsage

The ActiveBaseUsage property specifies whether to use the superclass specified by the ActiveBase property as the superclass for all threads.

Default = Cleared

ActiveDestructorGuard

The ActiveDestructorGuard property specifies the macro that starts protection for an active user object destructor. Default = START_DTOR_THREAD_GUARDED_SECTION

ActiveExecuteOperationName

The ActiveExecuteOperationName property sets the user object virtual table for an active object and passes it to a task in the task initialization function (RiCTask_init()). Follow these steps:

- Create a method with the following signature: struct RiCReactive * operation name (RiCTask * const)
- Set the operation name in the ActiveExecuteOperationName property.

- Start the execution of the active object task by calling the RICTASK_START() macro on the object.

The virtual function table member name is stored in the ActiveVtblName property.

Default = Empty string

ActiveGuardInitialization

The ActiveGuardInitialization property specifies the call that makes the active object event dispatching guarded.

Default = SetToGuardThread

ActiveIncludeFiles

The ActiveIncludeFiles property specifies the base class for threads when using selective framework includes. If a class is active and this property is defined, the file specified by the property is included in the class specification file.

ActiveInit

The ActiveInit property specifies the format of the declaration generated for the initializer for an active class.

Default = Empty string

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active objects, if the ActiveMessageQueueSize property for classes is left blank. The default value for the size of the message queue is language-dependent, as follows:

- C - The default value is RiCOSDefaultMessageQueueSize, which is a variable set in the implementation file of the OS adapter for a given operating system.
- C++ - The default value is OMOSThread::DefaultMessageQueueSize.
- Java - The default value is an empty string (blank).

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects, if the ActiveStackSize property for classes is left blank. The default value for the stack size is language-dependent, as follows:

- C - The default value is RiCOSDefaultStackSize, which is a variable set in the implementation file of the OS adapter for a given operating system.

ActiveThreadName

The ActiveThreadName property specifies the name of threads, if the ActiveThreadName property for classes is left blank. The default values are as follows:

- A string - Names the active thread.
- An empty string (blank) - The value is set in an operating system-specific manner, based on the value of the ActiveThreadName property for the framework.

Default = empty string (OS selects thread name)

ActiveThreadPriority

The ActiveThreadPriority priority specifies the priority of threads, if the ActiveThreadPriority property for classes is left blank. The default value for the priority of threads is language-dependent, as follows:

ActiveVtblName

The ActiveVtblName property stores the name of the virtual function table associated with a task (the RiCTask member of the structure).

Default = $objectName_activeVtbl

BooleanType

The BooleanType property specifies the Boolean type used by the framework.

Default = bool

CurrentEventId

The CurrentEventId property specifies the call or macro used to obtain the ID of the currently consumed event. Default = OM_CURRENT_EVENT_ID

DefaultProvidedInterfaceName

The DefaultProvidedInterfaceName property specifies the interface that must be implemented by the “in” part of a rapid port.

Default = DefaultProvidedInterface

DefaultReactivePortBase

The DefaultReactivePortBase property stores the base class for the generic rapid port (or default reactive

port). This base class relays all events.

Default = OMDefaultReactivePort

DefaultReactivePortIncludeFiles

The DefaultReactivePortIncludeFiles property specifies the include files that are referenced in the generated file that implements the class with the rapid ports.

Default = oxf/OMDefaultReactivePort.h

DefaultRequiredInterfaceName

The DefaultRequiredInterfaceName property specifies the interface that must be implemented by the “out” part of a rapid port.

Default = DefaultRequiredInterface

EnableDirectReactiveDeletion

The EnableDirectReactiveDeletion property specifies the call to the framework that supports direct deletion of reactive instances (using the delete operator) instead of graceful framework termination (using the reactive destroy() method). When using destroy(), the object waits in a zombie mode until all the events that are designated to it are removed from the active context queue, and then self -destructs. In this scheme, there is no need to traverse the queue of the active context to cancel pending events, and there is no need to make the reactive destructor guarded to ensure safe deletion. A reactive object can be either in a graceful termination or forced deletion (using the delete operator) state: you cannot use graceful deletion on an object that allows forced deletion, and vice versa. You can set a single reactive object in a forced deletion state, or set the entire system (all reactive instances) in a forced deletion state (as is done for backward compatibility). Graceful termination should not be used when a reactive part (of a composite class) runs in a context of an active object that is not part of, and different from, the composite active context. If you are using a Rational Rhapsody library component as part of an application where the main is not generated by Rational Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init(). If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call. Note that the property ADA_CG::Framework::UseDirectReactiveDeletion must be set to True for this property to take effect. When it is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init().

Default = OXF::supportExplicitReactiveDeletion();

EventBase

The EventBase property specifies the base class for all events, if the EventBaseUsage property is set to Checked.

Default = Empty string

EventBaseUsage

The EventBaseUsage property specifies whether to use the event superclass specified by the EventBase property as the parent of all events.

Default = Cleared

EventIncludeFiles

The EventIncludeFiles property specifies the base class for events when using selective framework includes. If events are defined in a package, the file specified by this property is included in the package specification file to enable the use of events and timeouts in the package. The default value for C is as follows: oxf/RiCEvent.h

EventSetParamsStatement

The EventSetParamsStatement property specifies a template for the body of the setParams() method, provided by the Rational Rhapsody framework for Java, to set the parameters of an event. For example, for an event of type evOn(), the default template would generate the following code in the body of the setParams() method: evOn params = (evOn) event; The default value is as follows: $eventType params = ($eventType) event;

FrameworkInitialization

The FrameworkInitialization property specifies the framework initialization code that is called by the main. The default value is as follows: OXF::initialize($(Argc)$(Argv)$(AnimationPortNumber) $(RemoteHost)$(TimerResolution)$(TimerMaxTimeouts) $(TimeModel))

HeaderFile

The HeaderFile property specifies the framework header files to be included in objects that are within the scope of a particular configuration.

Default = Empty string

To optimize your code for size, leave the HeaderFile property blank. In this way, you can explicitly include the framework only when needed.

IncludeHeaderFile

The IncludeHeaderFile property specifies whether to include the framework header files specified by the CG::Framework::HeaderFile property in the project.

Default = Cleared

InnerReactiveClassName

The InnerReactiveInstanceName property enables you to specify the name of a reactive class that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive. The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance.

Default = Reactive

InnerReactiveInstanceName

The InnerReactiveInstanceName property enables you to specify the name of a reactive instance that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive. The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance.

Default = reactive

InstrumentVtblName

The InstrumentVtblName property specifies the name of the virtual function table associated with animation objects. Each animated object has its own virtual function table (Vtbl). This table enables you to create your own framework, with its own virtual instrumentation functions, and connect it to Rational Rhapsody.

Default = $objectName_instrumentVtbl

IsCompletedCall

The IsCompletedCall property specifies the call or macro that determines whether the state reached a final state so it can be exited on a null transition. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name.

Default = IS_COMPLETED($State)

IsInCall

The IsInCall property specifies the query that determines whether the state is in the current active configuration. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name.

Default = IS_IN($State)

MakeFileName

The MakeFileName property enables you to specify a new name for the makefile. To use this property, add the following line to the .prp file:

Property MakeFileName String "MyFileName"

In this syntax, MyFileName specifies the name of the makefile.

NullTransitionId

The NullTransitionId property specifies the ID reserved for null transition consumption.

Default = OMEventNullId

OperationGuard

The OperationGuard property specifies the macro that guards an operation.

Default = GUARD_OPERATION

ProtectedBase

The ProtectedBase property specifies the base class for protected objects, if the ProtectedBaseUsage property is set to Checked.

Default = Empty string

ProtectedBaseUsage

The ProtectedBaseUsage property specifies whether to use the class specified by the ProtectedBase property as the base class for protected objects.

Default = Cleared

ProtectedClassDeclaration

The ProtectedClassDeclaration property affects how protected classes are implemented. Instead of inheriting from OMProtected, the class embeds an aggregate OMProtected. The aggregate member and helper methods are defined in the macro OMDECLARE_GUARDED (defined in omprotected.h).

Default = OMDECLARE_GUARDED

ProtectedIncludeFiles

The ProtectedIncludeFiles property specifies the base class for protected classes when using selective framework includes. The default value for C is as follows: oxf/RiCProtected.h

ProtectedInit

The ProtectedInit property specifies the declaration generated for the initializer for guarded objects.

Default = Empty string

ReactiveBase

The ReactiveBase property specifies the base class for all reactive classes, if the ReactiveBaseUsage property is set to Checked.

Default = Empty string

ReactiveBaseUsage

The ReactiveBaseUsage property specifies whether to use the class specified by the ReactiveBase property as the base class for all reactive objects.

Default = Cleared

ReactiveConsumeEventOperationName

The property ReactiveConsumeEventOperationName sets the user object virtual table for a reactive object. Follow these steps:

- Create a procedure with the following signature: procedure operation (a: in out RiAReactive; ev: in RiAEvent ev)
- Use the property ReactiveConsumeEventOperationName to set the operation name.

Rational Rhapsody Developer for Ada ignores all the values for the properties under the Framework metaclass except for this one.

Default = Blank

ReactiveCtorActiveArgDefaultValue

The ReactiveCtorActiveArgDefaultValue property specifies the default value of the active context argument in a reactive constructor.

Default = 0

ReactiveCtorActiveArgName

The ReactiveCtorActiveArgDefaultValue property specifies the name of the active context argument in a reactive constructor.

Default = activeContext

ReactiveCtorActiveArgType

The ReactiveCtorActiveArgDefaultValue property specifies the type of the active context argument in a reactive constructor.

*Default = IOxfActive**

ReactiveDestructorGuard

The ReactiveDestructorGuard property specifies the macro that starts protection of a section of code used for destruction of a reactive instance. This prevents a “race” (between the deletion and event dispatching) when deleting an active instance.

Default = START_DTOR_REACTIVE_GUARDED_SECTION

ReactiveEnableAccessEventData

The ReactiveEnableAccessEventData property specifies the code to be used to enable access to the specific event data in a transition (typically by assigning a local variable of the appropriate type). The property supports the $Event keyword so you can specify the event type.

Default = OMSETPARAMS($Event);

ReactiveGuardInitialization

The ReactiveDestructorGuard property specifies the framework call that makes the event consumption of a specific reactive class guarded.

Default = setToGuardReactive

ReactiveHandleEventNotConsumed

The ReactiveHandleEventNotConsumed property registers a method to handle unconsumed events in a reactive class. Specify the method name as this property’s value.

Default = Empty string

ReactiveHandleTONotConsumed

The ReactiveHandleTONotConsumed property registers a method to handle unconsumed trigger operations in a reactive class. Specify the method name as this property's value.

Default = Empty string

ReactiveIncludeFiles

The ReactiveIncludeFiles property specifies the base classes for reactive classes when using selective framework includes. If a class is reactive and this property is defined, the file specified by the property is included in the class specification file. For reactive classes, the header files specified by the following properties are also included:

- EventIncludeFiles - For the event base class
- ActiveIncludeFiles - If the class is guarded or instrumented

The default value for C is as follows: oxf/RiCReactive.h

ReactiveInit

The ReactiveInit property specifies the declaration for the initializer generated for reactive objects. The default pattern for C is as follows: $base_init($member, (void*)$mePtr, $task, $VtblName); The $base variable is replaced with the name of the reactive object during code generation. The string "_init" is appended to the object name in the name of the operation. For example, if the reactive object is named A, the initializer generated for A is named A_init(). The $member variable is replaced with the name of the reactive member (equivalent to the base class) of the object during code generation. The $mePtr variable is replaced with the name of the user object (the value of the Me property). The member and mePtr objects are not equivalent if the user object is active. The $VtblName variable is replaced with the name of the virtual function table for the object, specified by the ReactiveVtblName property.

Default = Empty string

ReactiveInterface

The ReactiveInterface property specifies the name of the interface class that forwards messages to an inner class instance of a reactive class in order to implement its reactive behavior.

Default = RiJStateConcept

ReactiveSetEventHandlingGuard

The ReactiveSetEventHandlingGuard property enables you to control the code generated within the constructor of a reactive class. When you use this property with guarded triggered operations, it enables guarding of the event handling (in order to provide mutual exclusion between the event and TO handling).

Default = setEventGuard(getGuard());

ReactiveSetTask

The ReactiveSetTask property specifies the string that tells a reactive object whether it is an active or a sequential instance.

Default = Empty string

ReactiveVtblName

The ReactiveVtblName property specifies the name of the virtual function table (Vtbl) associated with a reactive object. Each reactive object has its own Vtbl, which enables you to create your own framework and connect it to Rational Rhapsody.

Default = $objectName_reactiveVtbl

SetManagedTimeoutCanceling

The SetManagedTimeoutCanceling property is a property for backward compatibility that specifies whether the framework uses Rational Rhapsody version earlier than the 6.0 scheme of timeout creation and cancellation (where OMTimerManager is responsible for cancellation of timeouts) or the Rational Rhapsody 6.0 scheme. In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object). This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling). If you are using a Rational Rhapsody library component as part of an application where the main is not generated by Rational Rhapsody (for example, GUI applications), the framework initializes itself in full compatibility mode on the call to OXF::init(). If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call.

Default = OXF::setManagedTimeoutCanceling(true);

SetRhp5CompatibilityAPI

The SetRhp5CompatibilityAPI property specifies the call that configures models created before Rational Rhapsody 6.0 so they use the 5. x version of the framework instead of the new one. See UseRhp5CompatibilityAPI for more information on Version 5. x compatibility mode.

Default = OXF::setRhp5CompatibleAPI(true);

StaticMemoryIncludeFiles

The StaticMemoryIncludeFiles property specifies the files to be included in the package specification file if static memory management is enabled and you are using selective framework includes.

Default = oxf/MemAlloc.h

StaticMemoryPoolDeclaration

The StaticMemoryPoolDeclaration property specifies the declaration of the memory pool for timeouts. The default value follows:

DECLARE_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances)

StaticMemoryPoolImplementation

The StaticMemoryPoolImplementation property specifies the generated code in the implementation file for a memory pool implementation (see the BaseNumberOfInstances property). The default value is as follows:

IMPLEMENT_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances, $AdditionalNumberOfInstances, $ProtectStaticMemoryPool)

TestEventTypeCall

The TestEventTypeCall property specifies the test used in event consumption code to check if the currently consumed event is of a given type.

Default = IS_EVENT_TYPE_OF($Id)

TimeoutId

The TimeoutId property specifies the ID reserved for timeout events.

Default = OMTimeoutEventId

TimerMaxTimeouts

The TimerMaxTimeouts property specifies the maximum number of timeouts allowed simultaneously in the system, if the TimerMaxTimeouts property for the configuration is not overridden. In the framework, the default number of timers is 100.

Default = Empty string

TimerResolution

The TimerResolution property specifies the length of time that must pass until the timer should check for matured timeouts. In the framework, the default number of timers is 100.

Default = Empty string

UseDirectReactiveDeletion

The UseDirectReactiveDeletion property determines whether direct deletion of reactive instances (using the delete operator) is used instead of graceful framework termination (using the reactive destroy() method). When this property is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init(). See EnableDirectReactiveDeletion and the upgrade history on the support site for more information on this functionality.

Default = False

UseManagedTimeoutCanceling

The UseManagedTimeoutCanceling property specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (so OMTimerManager is responsible for cancellation of timeouts). The framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object). This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling). When loading a pre-6.0 model, Rational Rhapsody sets the project ADA_CG::Framework::UseManagedTimeoutCanceling to True to set the system-compatibility mode. See the upgrade history on the support site for more information.

Default = False

UseRhp5CompatibilityAPI

The UseRhp5CompatibilityAPI property specifies whether to use the virtual functions of the core implementation classes that existed in the pre-Rhapsody 6.0 framework. The Rational Rhapsody 6.0 framework introduces a set of interfaces for the core behavioral framework. The interfaces define a concise API for the framework and enable you to replace the actual implementation of these interfaces while maintaining the framework behavior. As a result of the interfaces' introduction, the framework behavioral classes (OMReactive, OMThread, and OMEvent) use a new set of virtual operations to implement the interfaces and provide the behavioral infrastructure. To support existing customizations of these classes (made by inheriting and overriding the virtual operations), the framework can work in a mode where the pre-6.0 API virtual operations are called. When loading a pre-6.0 model, Rational Rhapsody sets the project property ADA_CG::Framework::UseRhp5CompatibilityAPI to True to set the system-compatibility mode. If this is set to True, the pre-6.0 API is called by the framework instead of the interface-based API. Without this flag, user customizations will compile but will not be called. See the upgrade history on the support site for more information on the Version 5. x compatibility mode.

Default = False

Generalization

The Generalization metaclass contains a property used to support generalization.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = True

GNAT

This metaclass contains the properties that manipulate the GNAT operating system environment.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. This property reflects the change in Version 4.1 where the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags. The default values are as follows:

Environment Default Value INTEGRITY Empty string MultiWin32 IntegrityESTL ESTL

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches. The default values are as follows:

Environment Default Value INTEGRITY :optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 IntegrityESTL :optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 :cx_mode=extended_embedded :cx_lib=eece :stdcxxincdirs=$(INTEGRITY_ROOT)\eecxx :stdcxxincdirs=$(INTEGRITY_ROOT)\ansi MultiWin32 :cx_template_option=noimplicit :add_output_ext=checked :cx_e_option=msgnumbers :cx_option=exceptions :check=bounds :check=assignbound :check=nilderef :cx_template=local :cx_remark=14 :cx_remark=161 :cx_remark=837 :cx_remark=817 :cx_remark=815 :cx_remark=47 :cx_remark=69 :cx_remark=830 :cx_remark=550 :prelink.args=-r :prelink.args=-X7

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

Default =

:target_os=integrity :ada_library=full :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model. The default values are as follows:

Environment Default Value INTEGRITY :target_os=integrity IntegrityESTL MultiWin32 Empty MultiLine

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = sim800.

BriefErrorMessages

The BriefErrorMessages property determines whether a /brief option is generated on SPARK Examiner calls.

Default = Checked

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = "PENTIUM"

BSP_Libraries

The BSP_Libraries property specifies the default BSP libraries to link to. The default value is as follows:

"%RAVENROOT%/bsp/raven/standard_model" "%RAVENROOT%/bsp/system/simulator" "%RAVENROOT%/lib/extensions"

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation® CodeTest™.

Default = CXX=$AMC_HOME)\bin\ctcxx

COM

The COM property specifies whether the current component is a COM component. By default, this property is set to Checked for all COM components (stereotypes COM DLL, COM EXE, and COM TLB). If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows.

Default = Cleared

CompileCommand

The CompileCommand property is a string that enables you to specify a different compile command.

Default = Empty string

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

ConvertHostToIP

The ConvertHostToIP property specifies whether to convert the host name to an IP number. This is necessary because pSOSystem does not include a name service.

Default = Checked

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. The default values are as follows:

Environment Default Reserved Words MultiWin32 __asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance MicrosoftWinCE.NET NucleusPLUS-PPC PsosPPC PsosX86 Empty string

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default values are as follows:

Environment Default Compile Command Borland @echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) $(BCC32) -c @ $OMFileCPPCompileSwitches | -o$OMFileObjPath $OMFileImpPath INTEGRITYL echo Compiling $OMFileObjPath ... $(CPP) $OMFileCPPCompileSwitches -o "$OMFileObjPath" "$OMFileImpPath" IntegrityEST Linux @echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath MontaVista Microsoft $(CREATE_OBJ_DIR) $(CPP) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath" MicrosoftDLL MSStandardLibrary MicrosoftWinCE.NET $(CPP) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath" MultiWin32 $(COMPILEINFOLINE) $@... $(CPP) $OMFileCPPCompileSwitches -c "$OMFileImpPath" -o "$OMFileObjPath" NucleusPLUS-PPC $(CPP) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath OsePPCDiab Empty string OseSfk PsosPPC @echo Compiling $OMFileImpPath @$(CREATE_OBJ_DIR) @$(CXX) $(CXXOPTS) $OMFileCPPCompileSwitches $OMFileImpPath -o $OMFileObjPath PsosX86 @echo Compiling $OMFileImpPath @$(CREATE_OBJ_DIR) @$(CXX) $(CXXOPTS) $OMFileCPPCompileSwitches $OMFileImpPath -o $OMFileObjPath @$(INI) $(INIFLAG) $OMFileObjPath QNXNeutrinoCW @echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath QNXNeutrinoGCC Solaris2 @echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath Solaris2GNU VxWorks @echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = Empty string

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

CPPCompileSwitches

The CPPCompileSwitches property specifies the compiler switches. The default values are as follows:

Environment Default Compile Switches Borland -I$(BCROOT)\INCLUDE;.;"$(OMROOT)\LangCpp";

"$(OMROOT)\LangCpp\oxf";"$(OMROOT)\LangCpp\omCom";
-D_RTLDLL;_AFXDLL;WIN32;_CONSOLE;_MBCS;_WINDOWS; BORLAND;_BOOLEAN
$(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) $OMCPPCompileCommandSet -c Linux
-I. -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS)
$(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c
Microsoft /I . /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX
$OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D
"_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MicrosoftDLL
MicrosoftWin CE /I . /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX- /D
_WIN32_WCE=$(CEVersion) /D "$(CEConfigName)" $(MACHINE_CPP_FLAGS) /D
"_OM_NO_IOSTREAM" /D UNDER_CE=$(CEVersion) /D "UNICODE" /D
"_OM_UNICODE_ONLY" $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32"
$(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MSStandardLibrary /I . /I
$(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet
/D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" /D "OM_USE_STL"
$(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c OsePPCDiab -I.
-I$(OMROOT)$/LangCpp $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) OseSfk
PsosPPC $OMCPPCompileCommandSet PsosX86 QNXNeutrinoGCC -I. -I$(OMROOT)
-I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH)
$(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c Solaris2 Solaris2GNU
VxWorks -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks
$(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPU

The CPU property is a string that specifies the CPU type. The default values are as follows:

Environment Dependency Rule MicrosoftWinCE.NET x86 MontaVista 586

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

DEFExtension

The DEFExtension property is a string that specifies the extension for DLL definition files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .def

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the

makefile.

Default = Empty string

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DllExtension

The DllExtension property is a string that specifies the extension for DLL files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .dll

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = Empty string

See also the definition of the EntryPointDeclarationModifier property for more information.

EntryPointDeclarationModifier

The EntryPointDeclarationModifier property specifies a modifier for the entry point declaration. This property allows generation of the main() function in the specified syntax. To modify the main() signature implemented in the OSE adapter, do the following:

- Add the property EntryPointDeclarationModifier to your environment properties and set it to the main return value and name. For example: "int main"
- Set the EntryPoint property to the main arguments. For example: "int a, long b, char**"
- Generate the code.

You will get the following main() declaration: int main(int a, long b, char** c) { ... }

Default = OS_PROCESS

EnvironmentVarName

The EnvironmentVarName property specifies the name of the global variable that you must define in order to use the Embedded C++ compiler. It is used by the MultiMakefileGenerator. The value replaces the EnvironmentVarName value> keyword inside the property value BLDAdditionalOptions.

Default = INTEGRITY_ROOT

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ESTLCompliance

The ESTLCompliance property is a Boolean value that determines whether you are using the Embedded C++ (ESTL) environment and conform to its requirements. In instrumentation mode, the Rational Rhapsody code generator usually creates an OMAnimated"UserClass" friend class for each user-defined class. This class inherits from AOMInstance, if its "User Class" does not inherit from another class in the model. This inheritance is virtual and is needed for multiple inheritance support. Because ESTL does not support multiple inheritance (as far as virtual inheritance), the Rational Rhapsody Developer for C++ code generator will not create “virtual” inheritance if ESTLCompliance is set to Checked.

To support ESTL compliance, Rational Rhapsody includes a new check to recognize the following elements of ESTL-noncompliance:

- Multiple inheritance, caused by the user model (several superclasses)
- Multiple inheritance, caused by Rational Rhapsody (an active reactive class is generated with two base classes: OMReactive and OMThread)
- Multiple inheritance, caused by a combination of the following factors:
- An active class containing a superclass
- A reactive class containing a superclass

Virtual inheritance, declared by the user in the features of the superclass

In these cases, Rational Rhapsody displays the following warning message for each problematic class: "ESTL does not support multiple/virtual inheritance" Note that this check runs only when the ESTLCompliance property is set to Checked.

Default = Checked

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .exe

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = Empty string

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application. The default values are as follows:

Environment Default Reserved Words INTEGRITY Integrity IntegrityESTL IntegrityESTL MultiWin32 MultiWin32

GeneratedAllDependencyRule

The GeneratedAllDependencyRule property specifies whether to automatically generate the “all:” rule as part of the expansion of the $OMContextMacros keyword in the makefile. If this is Cleared, you can define the makefile macros manually.

Default = Cleared

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries). The default values are as follows:

Environment Default Value Borland "$(OMROOT)\LangCpp\lib\bc5WebComponents.lib", "$(OMROOT)\lib\bc5WebServices.lib", -lsocket INTEGRITY "$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT)" IntegrityESTL Linux $(OMROOT)/LangCpp/lib/linuxWebComponents$(LIB_EXT), $(OMROOT)/lib/linuxWebServices$(LIB_EXT) Microsoft $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)

WebComponents $(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT),ws2_32$(LIB_EXT) MicrosoftWinCE.NET $(OMROOT)\LangCpp\lib\$(LIB_PREFIX) WebComponents $(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT), winsock.lib MontaVista $(OMROOT)/LangCpp/lib/$(LIB_PREFIX)WebComponents$(CPU) $(LIB_EXT), $(OMROOT)/lib/$(LIB_PREFIX)WebServices(CPU)(LIB_EXT) NucleusPLUS-PPC $(OMROOT)\LangCpp\lib\$(LIB_PREFIX) WebComponents $(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT) QNXNeutrinoCW $(OMROOT)\LangCpp\lib\ QNXCWWebComponents (CPU)(CPU_SUFFIX)$(LIB_EXT), $(OMROOT)\lib\QNXCWWebServices$(CPU)$(CPU_SUFFIX)$(LIB_EXT), -lsocket QNXNeutrinoGCC $(OMROOT)\LangCpp\lib\QNXWebComponents$(LIB_EXT), $(OMROOT)\lib\ QNXWebServices $(LIB_EXT), -lsocket Solaris2 $(OMROOT)\LangCpp\lib\sol2WebComponents$(LIB_EXT),$(OMROOT)\lib\sol2WebServices$(LIB_EXT), -lsocket -lnsl Solaris2GNU $(OMROOT)\LangCpp\lib\ sol2WebComponentsGNU $(LIB_EXT), $(OMROOT)\lib\sol2WebServicesGNU$(LIB_EXT),-lsocket -lnsl VxWorks $(OMROOT)\LangCpp\lib\vxWebComponents$ CPU)$(LIB_EXT), $(OMROOT)\lib\ vxWebServices (CPU)(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default value for QNXNeutrinoCW is Cleared; for the other environments, the default value is Checked.

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The default values are as follows:

Environment Default Value QNXNeutrinoCW $OMROOT/DLLs/CodeWarriorIDE.dll INTEGRITY Empty string IntegrityESTL VxWorks $OMROOT/DLLs/TornadoIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = Empty string

InvokeCodeGeneration

The InvokeCodeGeneration property specifies the command line used by Rational Rhapsody to run an external code generator. The generator should implement the IRPExternalCodeGenerator connection point. To use an external code generator, you need the appropriate license; an external generator license is part of the Rational Rhapsody Developer for Ada package license. The default values are as follows:

Metaclass Default Value GNAT "$OMROOT/etc/Executer.exe" "$OMROOT/etc/invokeScriptor.bat" MultiWin32 OBJECTADA RAVEN_PPC SPARK INTEGRITY "$OMROOT/etc/Executer.exe" "$OMROOT\etc\IntegrityAdaMake.bat $makefile $maketarget"

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default =

"$OMROOT/etc/Executer.exe" "$OMROOT\etc\GnatMake.bat $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored. The default values are as follows:

Environment Default Value INTEGRITY $OMROOT/etc/MultiMakefileGenerator.exe MultiWin32 IntegrityESTL $OMROOT/etc/IntegrityMakefileGenerator.bat All others Empty string

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MainIncludes

The MainIncludes property is a string that specifies the files that need to be included in the main program generated for an application.

Default = ose.h

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bat

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

Default = Empty MultiLine

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib

LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody-generated macros in the makefile. For example: ################ Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT CPP_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody-generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody-generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies

$OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions

The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows:

################## Linking instructions ##################
##
$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = Empty string

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OMCPU

The OMCPU property is resolved in the MakeFileContent property as the CPU type. The QNXNeutrinoCW environment uses the custom keywords feature to enable you to select the CPU without modifying the makefile template.

Default = x86

OMCPU_SUFFIX

The OMCPU_SUFFIX property is resolved in the MakeFileContent property as the CPU extension (which is required for PPC targets). The QNXNeutrinoCW environment uses the custom keywords feature to enable you to select the CPU without modifying the makefile template.

Default = ($NO_CPU_EXT)

OpenHTMLReports

The OpenHTMLReports property specifies whether to open the HTML reports when the examination is complete.

Default = True

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation. The default values are as follows:

Environment Default Value Borland / Linux MontaVista MultiWin32 PsosPPC PsosX86 QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU JDK \ Microsoft MicrosoftDLL MSStandardLibrary

ProcessToKillAtStopExec

The ProcessToKillAtStopExec property stops the running process of the Java application when you select Code > Stop Execution in the Rational Rhapsody GUI.

Default = "Java"

QuoteOMROOT

The QuoteOMRoot property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource file. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The default values are as follows:

Environment Default Value Microsoft $(RC) /Fo"$(TARGET_MAIN).res" $(TARGET_MAIN)$OMRCExtension MicrosoftDLL MultiWin32 Empty string

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .rc

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change. The default values are as follows:

Environment Default Value Borland -DOM_REUSABLE_STATECHART_IMPLEMENTATION Linux NucleusPLUS-PPC OsePPCDiab OseSfk PsosPPC QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks Microsoft /D "OM_REUSABLE_STATECHART_ IMPLEMENTATION" MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary INTEGRITY OM_REUSABLE_STATECHART_IMPLEMENTATION IntegrityESTL MontaVista MultiWin32

RPFrameWorkDll

The RPFrameWorkDll property determines whether the configuration uses the DLL flavor of the framework libraries. To use OXF DLLs for the creation of COM ATL components, you must set this property to Checked before you generate code. Rational Rhapsody COM ATL components use a DLL version of the OXF. This version of the OXF allows the use of multiple Rational Rhapsody-generated DLL/executable components confined to a single process. There are three versions of the OXF DLL:

DLL Version Animation Enabled Trace Enabled oxfdll.dll No No oxfanimdll.dll Yes No oxftracedll.dll No Yes

OXF DLLs are not a part of a typical Rational Rhapsody installation, and must be built from the OXF C++ sources. To obtain these sources, you can either perform a custom installation or update the install to add the sources to your existing installation. To rebuild the framework DLLs, run the following in your $OMROOT\LangCpp folder: nmake -f msoxfanimtracedll.mak CFG=oxfdll nmake -f msoxfanimtracedll.mak CFG=oxfanimdll nmake -f msoxfanimtracedll.mak CFG=oxftracedll

To use OXF DLLS for the creation of COM ATL components, set the following component configuration properties to True before you generate code:

- ADA_CG::Microsoft::RPFrameWorkDll
- ADA_CG::MicrosoftDLL::RPFrameWorkDll this is True by default

In addition, make sure that the following are included in the system environment path:

- OXF DLL path ($OMROOT\LangCpp\lib)
- The full path to regsrv32.exe

Without these settings, COM ATL components are not registered and cannot run. Limitations:

- Rational Rhapsody components with different instrumentation settings (both Animation and Tracing) are not supported within a single process. However, you can mix instrumented and noninstrumented DLLs, as long as you use the instrumented DLL.
- Mixing Rational Rhapsody components that link to the DLL and the library version of OXF is not supported within a single process.

Default = False

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

SpecFilesInDependencyRules

The SpecFilesInDependencyRules property specifies whether to include specification files in makefile dependency rules. The OSE makefile does not support specification files in the Dependency line. Therefore, the default for OSE is False. When this property is False, no .h files are added to the Dependency line of the makefile. The default value for GNAT is True; for OSE, the default value is False.

SubSystem

The SubSystem property (ADA_CG::Microsoft/MicrosoftDLL/MultiWin32) is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

Default = /SUBSYSTEM:console

TargetConfigurationFileName

The TargetConfigurationFileName property specifies the name of the target configuration file to be passed as an argument to the SPARK Examiner.

Default = Empty string

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style.

If this property is set to False, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = True

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIX-style path names. The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

The default value for the following environments is Checked:

Linux MontaVista PsosPPC PsosX86 QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU

The default value for the following environments is Cleared:

JDK OsePPCDia OseSfk

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuratGenerate Code For Actors checkmark (located in the configuration Initialization tab).

Default = False

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

GNATVxWorks

This metaclass contains the properties that manipulate the GNATVxWorks operating system environment.

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser.

Important: Do NOT change it using the Properties window or by modifying the site.prp file!

This property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileCommand

The CompileCommand property is a string that enables you to specify a different compile command.

Default = Empty string

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = Empty string

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = Empty string

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = Empty string

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .exe

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = Empty string

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\GnatVxWMake.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bat

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

Default = Empty MultiLine

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody-generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords

separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT CPP_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody-generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody-generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions

The linking instructions section

of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows:

################## Linking instructions ##################
##
$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = Empty string

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

QuoteOMROOT

The QuoteOMRoot property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

INTEGRITY

This metaclass contains the properties that manipulate the INTERGRITY operating system environment.

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

:optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

Default =

:target_os=integrity :ada_library=full :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default = :target_os=integrity

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = sim800

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = Default

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .mod

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\IntegrityMake.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bld

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([a-zA-Z][:][^:,]+)(,|: Error:|: Warning:) line ([0-9]+)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround

if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

INTEGRITY5

This metaclass contains the properties that manipulate the INTERGRITY5 operating system environment.

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

-Ospace --diag_suppress 14,550

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

Default =

--ada_library_full -dynamic -non_shared

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default = --ada_library_full -dynamic -non_shared

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = sim800

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = --no_debug

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .mod

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\Integrity5Make.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .gpj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([a-zA-Z][:][^:,]+)(,|: Error:|: Warning:) line ([0-9]+)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

Multi4Win32

This metaclass contains the properties that manipulate the Multi4Win32 operating system environment.

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

Default =

-threading=multiple --ada_library_full

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default = Empty MultiLine

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty MultiLine

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = --no_debug

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .exe

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from

the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\Multi4Win32Make.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .gpj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([a-zA-Z][:][^:,]+)(,|: Error:|: Warning:) line ([0-9]+)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

MultiWin32

This metaclass contains the properties that manipulate the MultiWin32 operating system environment.

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

Default =

:win32_threading=multiple :ada_library=full

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default = Empty MultiLine

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = Default

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .exe

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeCodeGeneration

The InvokeCodeGeneration property specifies the command line used by Rational Rhapsody to run an external code generator. The generator should implement the IRPExternalCodeGenerator connection

point. To use an external code generator, you need the appropriate license; an external generator license is part of the Rational Rhapsody Developer for Ada package license.

Default = "$OMROOT/etc/Executer.exe" "$OMROOT/etc/invokeScriptor.bat"

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default =

"$OMROOT/etc/Executer.exe" "$OMROOT\etc\MultiWin32Make.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bld

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([a-zA-Z][:][^:,]+)(,|: Error:|: Warning:) line ([0-9]+)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

OBJECTADA

This metaclass contains the properties that manipulate the OBJECTADA operating system environment.

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = -ga

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .exe

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeCodeGeneration

The InvokeCodeGeneration property specifies the command line used by Rational Rhapsody to run an external code generator. The generator should implement the IRPExternalCodeGenerator connection point. To use an external code generator, you need the appropriate license; an external generator license is part of the Rational Rhapsody Developer for Ada package license. The default values are as follows:

Metaclass Default Value GNAT "$OMROOT/etc/Executer.exe" "$OMROOT/etc/invokeScriptor.bat" MultiWin32 OBJECTADA RAVEN_PPC SPARK INTEGRITY "$OMROOT/etc/Executer.exe" "$OMROOT\etc\IntegrityAdaMake.bat $makefile $maketarget"

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaMake.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bat

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:] (Warning|Error)[:] line ([0-9]+)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

Operation

The Operation metaclass contains properties that control operations.

ActivityReferenceToAttributes

The ActivityReferenceToAttributes property specifies whether Rational Rhapsody should generate references in the functor object, thereby giving you direct access to the attributes of the class that owns the modeled operation (without the need for this_).

Default = True

AnimAllowInvocation

The AnimAllowInvocation property specifies whether primitive and triggered operations can be called during instrumentation. If an operation is called during animation, its return value is displayed in the output window; if it is traced, the return value is displayed in the console. The possible values are as follows:

- All - Enable all operation calls, regardless of visibility.
- None - Do not enable operation calls.
- Public - Enable calls to public operations only.
- Protected - Enable calls to protected operations only.

Default = None

DeclarationPosition

The DeclarationPosition property specifies where the type declaration appears. The possible values are as follows:

- BeforeClassRecord - The type declaration appears before the class record (CR) declaration if CR has a visibility set to public, and before the class record forward declaration if CR has a visibility set to private.
- AfterClassRecord - The type declaration appears after the class record declaration if CR has a visibility set to public, and after the class record forward declaration if CR has a visibility set to private.
- StartOfDeclaration - The type declaration appears among the first declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the first declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.
- EndOfDeclaration - The type declaration appears among the last declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the last declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.

If the ADA_CG::Type::Visibility property is set to "Body," no matter the settings of ADA_CG::Type::DeclarationPosition property, the type declaration still appears in the package body.

Default = AfterClassRecord

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

Tag - The value of the specified the element tag Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

EntryCondition

The EntryCondition property specifies the task guard.

Default = Empty string

GenerateImplementation

The GenerateImplementation property specifies whether to generate the body for the operation. To generate Import pragmas in Rational Rhapsody Developer for Ada, set this property to Cleared and add the "pragma..." declaration in the Ada_CG::Operation::SpecificationEpilog property.

Default = Checked

ImplementActivityDiagram

The ImplementActivity Diagram property enables or disables code generation for activity diagrams.

Default = False

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationName

The ImplementationName property enables you to give an operation one model name and generate it with another name. It is introduced as a workaround that enables you to generate const and non-const operations with the same name. For example:

- Create a class A.
- Add a non-const operation f().
- Add a const operation f_const().
- Set the ADA_CG::Operation::ImplementationName property for f_const() to “f.”
- Generate the code.

The resulting code is as follows: class A { ... void f(); /* the non const f */ ... void f() const; /* actually f_const() */ ... }; The creation of two operations with the same signature, differing only in whether it is a const, is a common practice in C++, especially for STL users.

Default = Empty string

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for Ada If the operation is marked as inline, the package specification includes an inline pragma after the declaration. For example: pragma inline (operation name);

The two possible values for Rational Rhapsody Developer for Ada are as follows:

- none (default)
- use_pragma

IsAnimationHelper

The IsAnimationHelper property indicates whether the operation should be generated only when animating the model.

Default = Cleared

IsEntry

The IsEntry property indicates whether the operation is a task entry or a regular operation in "AdaTask" and "AdaTaskType" classes.

Default = Cleared

IsExplicit

The boolean property IsExplicit allows you to specify that a constructor is an explicit constructor.

Default = False

IsNative

The IsNative property specifies whether the Java modifier “native” should be added to an operation in the source file. The body of such operations, if specified, is ignored by the code generator.

Default = False

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.

- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation.

Default = Empty string

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

Me

The Me property specifies the name of the first argument to operations generated in C.

Default = me

MeDeclType

The MeDeclType property is a string that specifies the type of the first argument to operations generated in C, as a pointer to an object or object type. The default value is as follows: $objectName* const

The variable $objectName is replaced with the name of the object or object type.

PrivateQualifier

The PrivateQualifier property specifies the qualifier that is printed at the beginning of a private operation declaration or definition. You can set this property to an empty string to prevent the generation of the static qualifier in the private function declaration or definition.

Default = static

ProtectedName

The ProtectedName property specifies the pattern used to generate names of private operations in C. The default value is as follows: $opName The $opName variable specifies the name of the operation. For example, the generated name of a private operation go() of an object A is generated as: go()

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C. The default value is as follows: $objectName_$opName The $objectName variable specifies the name of the object; the $opName variable specifies the name of the operation. For example, the generated name of a public operation go() of an object A is generated as: A_go()

PublicQualifier

The PublicQualifier property specifies the qualifier that is printed at the beginning of a public operation declaration or definition. Note that the Static checkmark in the operation dialog UI is disabled in Rational Rhapsody Developer for C because the checkmark is associated with class-wide semantics that are not supported by Rational Rhapsody Developer for C. When loading models from previous versions, the Static check box is cleared; if the operation is public, the C_CG::Operation::PublicQualifier property value is set to Static in order to generate the same code.

Default = Empty string

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.

- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

RenamesKind

The RenamesKind property specifies whether the renaming of the operation designated in the ADA_CG::Operation::Renames property is "as specification" or "as body."

Default = Specification

ReturnTypeByAccess

The ReturnTypeByAccess property determines whether the return type is generated as an access type or a regular type. Note that this property is applicable only to classes for which an access type is generated.

Default = None

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

TaskDefaultScheme

The TaskDefaultScheme property sets the task default entry scheme. The possible values are as follows:

- Conditional
- Timed
- None

Default = None

TaskDefaultSchemeDelayStatement

The TaskDefaultScheme property sets the task default entry statement for timed entry schemes.

Default = Empty MultiLine

ThisByAccess

The ThisByAccess property specifies whether to pass the this parameter as an access mode parameter for a non-static operation.

Default = Cleared

ThisName

The ThisName property enables you to specify the name of the this parameter, which specifies the instance.

Default = this

ThrowExceptions

The ThrowExceptions property specifies the exceptions that an operation can throw. Separate multiple

exceptions with commas.

Default = Empty string

VirtualMethodGenerationScheme

The VirtualMethodGenerationScheme property enables backward-compatibility mode for methods of interface and abstract classes. The possible values are as follows:

- Default - The class type is class-wide, but the this parameters are not.
- ClassWideOperations - The class type is not class-wide, but the this parameters are.

Default = Default

Package

The Package metaclass contains properties that affect packages.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = True

ContributesToNamespace

The ContributesToNamespace property specifies whether the packages contained in this package is declared as children packages of this package. Regardless of the setting, a directory is created for the current package to hold its contained elements.

Default = AsGeneratePackageCode

To generate the class name without a package name, set to False. This will remove the package name prefix from the class name.

DeclarationPosition

The DeclarationPosition property enables you to control the declaration order of attributes. The possible values are as follows:

- Default - Similar to the AfterClassRecord setting, with the following difference:
- For static attributes defined in a class with the property ADA_CG::Attribute::Visibility set to Public, these attributes are generated after types whose ADA_CG::Type::Visibility property is set to Public.
- You should not use this setting for new models.
- BeforeClassRecord - Generate the attribute immediately before the class record.
- AfterClassRecord - Generate the attribute immediately after the class record.
- StartOfDeclaration - Generate the attribute immediately after the start of the section (private or public part of the specification, or package body).
- EndOfDeclaration - Generate the attribute immediately before the end of the section (private or public part of the specification, or package body).

Default = EndOfDeclaration

DefineNameSpace

The DefineNameSpace property specifies whether a package defines a namespace. A namespace is a declarative region that attaches an additional identifier to any names declared inside it.

Default = False

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

Tag - The value of the specified the element tag Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

EventsBaseID

The EventsBaseID property specifies the base ID for events.

Default = 1

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces.

Default = Empty string

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body.

Default = Empty MultiLine

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body.

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

InitializationCode

The InitializationCode property specifies the user-defined initialization code to add to the package body.

Default = Empty MultiLine

IsNested

The IsNested property specifies whether to generate the class or package as nested.

Default = Cleared

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private.

Default = Cleared

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package.

Default = Public

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = Auto

PackageClassNamePolicy

The PackageClassNamePolicy property specifies the naming policy for classes generated by Rational Rhapsody. The software generates a class for each package in the Rational Rhapsody Developer for Java model. The possible values are as follows:

- Default - Use the default naming style (the package class name is the same as the package name).
- WithSuffix - Add a suffix to the class name. The suffix is “_pkgClass."

Default = Default

PackageEventIdRange

The PackageEventIdRange property specifies the maximum number of events allowed in a package. This property is set on the component level.

Default = 200

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a "renames" dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification.

Default = Empty MultiLine

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification.

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract .” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces.

Default = Empty string

Port

The Port metaclass controls whether code is generated for ports.

Generate

The Generate property specifies whether to generate code for a particular type of element.

Default = Checked

RAVEN_PPC

This metaclass contains the properties that manipulate the RAVEN_PPC operating system environment.

BSP_Libraries

The BSP_Libraries property specifies the default BSP libraries to link to.

Default =

"%RAVENROOT%/bsp/raven/standard_model" "%RAVENROOT%/bsp/system/simulator" "%RAVENROOT%/lib/extensions"

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = -ga

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .x

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaRavenPPCMake.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bat

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:] (Warning|Error)[:] line ([0-9]+)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network

by default in a given environment.

Default = Cleared

Relation

The Relation metaclass contains properties that affect relations.

Add

The Add property specifies the command used to add an item to a container.

Default = Add_$target:c

AddGenerate

This property specifies whether to generate an Add() operation for relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = True

Clear

The Clear property specifies the name of an operation that removes all items from a relation.

Default = Clear_$target:c

ClearGenerate

This property specifies whether to generate a Clear() operation for relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

CreateComponent

The CreateComponent property specifies the name of an operation that creates a new component in a composite class. Default = New_$target:c

CreateComponentGenerate

This property specifies whether to generate a CreateComponent operation for composite objects. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

DataMemberVisibility

The DataMemberVisibility property specifies the visibility of the relation data member. For example, if the relation is implemented as a pointer, this property determines whether the pointer data member is declared as public, private, or protected. The default value for Ada and C is Private; the default value for C++ and Java is Protected.

DeleteComponent

The DeleteComponent property specifies the name of an operation that deletes a component from a composite class.

Default = Delete_$target:c

DeleteComponentGenerate

This property specifies whether to generate a DeleteComponent() operation for composite objects. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation

rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument's description Operation Primitive operations, triggered operations, $Arguments - The operation argument's description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

Tag - The value of the specified the element tag Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

Find

The Find property specifies the name of an operation that locates an item among relational objects.

Default = Find_$target:c

FindGenerate

This property specifies whether to generate a Find() operation for relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Cleared

Get

The Get property specifies the name of an operation that retrieves the relation currently pointed to by the

iterator.

Default = Get_$target:c

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index. The ContainerTypes>RelationtypeGetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

Default =Get_$target:c.

GetAtGenerate

The GetAtGenerate property specifies whether to generate a getAt() operation for relations. The possible values are as follows:

- Checked - Generate a getAt() operation for relations.
- Cleared - Do not generate a getAt() operation for relations. Setting the GetAtGenerate property to False is one way to optimize your code for size.

Default = Cleared

GetEnd

The GetEnd property specifies the name of an operation that points the iterator to the last item in a collection.

Default = Get_$target:cEnd

GetEndGenerate

The GetEndGenerate property specifies whether to generate a GetEnd() operation for relations.

Default = Checked

GetGenerate

The GetGenerate property specifies whether to generate accessor operations for relations.

Default = Checked

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key. For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)Default = get$cname:c

GetKeyGenerate

The GetKeyGenerate property specifies whether to generate getKey() operations for relations. Setting this property to False is one way to optimize your code for size.

Default = True

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

ImplementWithStaticArray

The ImplementWithStaticArray property specifies whether to implement relations as static arrays. The possible values are as follows:

- Default - Rational Rhapsody provides the appropriate implementation for all fixed and bounded relations.
- FixedAndBounded - All fixed and bounded relations are generated into static arrays.

To generate C-like code in C++ or Java, modify the value of the ImplementWithStaticArray property to FixedAndBounded.

Default = FixedAndBounded

InitializeComposition

The InitializeComposition property controls how a composition relation is initialized. The possible values are as follows:

- InInitializer
- InRecordType
- None

Default = InInitializer

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for Ada If the operation is marked as inline, the package specification includes an inline pragma after the declaration. For example: pragma inline (operation name);

The two possible values for Rational Rhapsody Developer for Ada are as follows:

- none
- use_pragma

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased.

Default = Cleared

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

ObjectInitialization

The ObjectInitialization property specifies what kind of initialization will occur for the initial instances of a configuration. The possible values are as follows:

- Full - Instances are initialized and their behavior is started.
- Creation - Instances are initialized but their behavior is not started.
- None - Instances are not initialized and their behavior is not started.

Default = Full

Remove

The Remove property specifies the name of an operation that removes an item from a relation.

Default = Remove_$target:c

RemoveGenerate

This property specifies whether to generate a Remove() operation for relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)Default = remove$cname:c

RemoveKeyGenerate

The RemoveKeyGenerate property specifies whether to generate a removeKey() operation for qualified relations. Setting this property to False is one way to optimize your code for size.

Default = True

RemoveKeyHelpersGenerate

The RemoveKeyHelpersGenerate property enables you to control the generation of the relation helper methods (for example, _removeItsX() and __removeItsX()). The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the ADA_CG::Relation::RemoveKey property.

Default = True

SafeInitScalar

The SafeInitScalar property specifies whether to initialize scalar relations as null pointers.

Default = False

Set

The Set property specifies the name of the mutator generated for scalar relations.

Default = Set_$target:c

SetGenerate

This property specifies whether to generate mutators for relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

SpecificationEpilog

The SpecificationEpilog property enables you to add code to the end of the declaration of a model element (a configuration). This property enables you to wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes Yes Outside Package Yes Yes Inside

Default = Empty MultiLine

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside of Namespace? Class Yes No Inside Package Yes Yes Inside

Empty MultiLine

Static

The Static property is a Boolean value that determines whether class-wide relations are enabled. Class-wide members of a class are shared between all instances of that class and are mapped as static. When a relation is tagged as static:

- The data member is generated as static (with the static keyword).
- The relation accessors are generated as static.
- The mutators of directional relations are generated as static. The mutators of symmetric relations are generated as common (non-static) operations.

Note the following behavior and restrictions:

- If there are links between instances based on static relations, code generation will initialize all the valid links. In case of a limited relation size, the last initialization is preserved.
- When you generate instrumented code (animation or tracing), relation NOTIFY calls are not added to static relation mutators.
- Animation associates static relations with the class instances, not the class itself.
- In an instrumented application (animation or tracing), the static relations names appear in each instance node; however, the values of directional static relations are not visible.

See also the properties CG::Relation::Containment, Containertype::Relationtype::CreateStatic, and Containertype::Relationtype::InitStatic.

Default = False

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language.

Default = Public

SPARK

This metaclass contains the properties that manipulate the SPARK operating system environment.

BriefErrorMessages

The BriefErrorMessages property determines whether a /brief option is generated on SPARK Examiner calls.

Default = Checked

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty string

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches.

Default = Empty string

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .exe

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .adb

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "$OMROOT\etc\SPARKMake.bat $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bat

OpenHTMLReports

The OpenHTMLReports property specifies whether to open the HTML reports when the examination is complete.

Default = Checked

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .ads

TargetConfigurationFileName

The TargetConfigurationFileName property specifies the name of the target configuration file to be passed as an argument to the SPARK Examiner.

Default = Empty string

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

Type

The Type metaclass contains a property that affects the visibility of data types.

AccessTypeUsage

This property defines the access type.

Default = None

AnimEnumerationTypeImage

This property is a Boolean value that determines whether the Image attribute is used for enumerated types when using animation.

Default = Cleared

AnimSerializeOperation

The AnimSerializeOperation property enables you to specify the name of an external function used to animate all attributes and arguments that are of that type. Rational Rhapsody can animate (display) the values of simple types and one-dimensional arrays without any problem.

To display the current values of such attributes during an animation session, run the Features window for the instance. However, if you want to animate a more complex type, such as a date, the type must be converted to a string (char *) for Rational Rhapsody to display it. This is generally done by writing a global function, an instrumentation function, that takes one argument of the type you want to display, and returns a char *.

You must disable animation of the instrumentation function itself (using the Animate and AnimateArguments properties for the function). For example, you can have a type tDate, defined as follows: typedef struct date { int day; int month; int year; } %s; You can have an object with an attribute

count of type int, and an attribute date of type tDate. The object can have an initializer with the following body: me-date.month = 5; me-date.day = 12; me-date.year = 2000;

If you want to animate the date attribute, the AnimSerializeOperation property for date must be set to the name of a function that will convert the type tDate to char *. For example, you can set the property to a function named showDate. This function name must be entered without any parentheses. It must take an attribute of type tDate and return a char *. The Animate and AnimateArguments properties for the showDate function must be set to False. The implementation of the showDate function might be as follows: showDate(tDate aDate) { char* buff; buff = (char*) malloc(sizeof(char) * 20); sprintf(buff,"%d %d %d", aDate.month,aDate.day,aDate.year); return buff; }

When you run this model with animation, instances of this object will display a value of 5 12 2000 for the date attribute in the browser. If the showDate function is defined in the same class that the attribute belongs to and the function is not static, the AnimSerializeOperation property value should be similar to the following:

myReal-showDate

This value shows that the function is called from the serializeAttributes function, located in the class OMAnimatedclassname. The showDate function must allocate memory for the returned string via the malloc/alloc/calloc function in C, or the new operator in C++. Otherwise, the system will crash.

Default = Empty string

AnimUnserializeOperation

The AnimUnserializeOperation property converts a string to the value of an element (the opposite of the AnimSerializeOperation property). Unserialize functions are used for event generation or operation invocation using the Animation toolbar to convert the string (received from the user) to the value of the event or operation before the event generation or operation invocation. For example, your serialization operation might look similar to the following:

char* myX2String(const Rec f) { char* cS = new char[OutputStringLength]; /* conversion from the Rec type to string */ return (cS); }

The unserialization operation would be: Rec myString2X (char* C, Rec T) { T = new Trc; /* conversion of the string C to the Rec type */ delete C; return (T); }

Default = Empty string

DeclarationPosition

The DeclarationPosition property specifies where the type declaration appears. The possible values are as follows:

- BeforeClassRecord - The type declaration appears before the class record (CR) declaration if CR has a visibility set to public, and before the class record forward declaration if CR has a visibility set to private.
- AfterClassRecord - The type declaration appears after the class record declaration if CR has a visibility set to public, and after the class record forward declaration if CR has a visibility set to private.

- StartOfDeclaration - The type declaration appears among the first declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the first declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.
- EndOfDeclaration - The type declaration appears among the last declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the last declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.

If the ADA_CG::Type::Visibility property is set to "Body," no matter the settings of ADA_CG::Type::DeclarationPosition property, the type declaration still appears in the package body.

Default = BeforeClassRecord

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description

Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified the element tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

Keyword names can be written in parentheses. For example:

$(Name)

If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the ADA_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

Default = Empty string

EnumerationAsTypedef

The EnumerationAsTypedef property specifies whether the generated enum should be wrapped by a typedef. This property is applicable to enumeration types in C and C++.

Default = True

Final

The Final property, when set to Cleared, specifies that the generated record for the class is a tagged record. This property applies to ADA95.

Default = Checked

In

The In property specifies how code is generated when the type is used with an argument that has the modifier "In."

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier "InOut."

IsLimited

The IsLimited property determines whether the class or record type is generated as limited.

Default = Cleared

LanguageMap

The LanguageMap property specifies the Ada declaration for Rational Rhapsody language-independent types.

Default = Empty string

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out."

PrivateName

The PrivateName property specifies the pattern used to generate names of private operations in C.

Default = $typeName

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C.

Default = $objectName_$typeName

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code.

Default = ""*

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type.

*Default = $type**

StructAsTypedef

The StructAsTypedef property specifies whether the generated struct should be wrapped by a typedef. This property is applicable to structure types in C and C++.

Default = True

TriggerArgument

The TriggerArgument property is used for mapping event and triggered operation arguments to code instead of the In, InOut, and Out properties. A different property is required because of code generation limitations related to event arguments. See also:

- In
- InOut
- Out

Default = $type

UnionAsTypedef

The UnionAsTypedef property specifies whether the generated union should be wrapped by a typedef. This property is applicable to union types in C and C++.

Default = True

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

Default = Public

Ada_ReverseEngineering

In addition to the ReverseEngineering subject, Rational Rhapsody provides language-specific subjects to control how Rational Rhapsody imports legacy code. Most of the properties are identical for each language. Any language-specific properties are clearly labeled. In general, most of the reverse engineering (RE) properties have graphical representation in the Reverse Engineering Options window. You should change the options using this window instead of the corresponding properties.

The metaclasses are as follows:

- ApproximatedConstructs
- Filtering
- ImplementationTrait
- Main
- MFC
- MSVC60
- Parser
- Promotions

ApproximatedConstructs

The ApproximatedConstructs metaclass contains properties that control inheritance issues in reverse engineering.

ProtectedInheritance

The ProtectedInheritance property specifies whether protected inheritance is modeled as public inheritance. The possible values are as follows:

- Property - The inheritance is imported to the PrivateInherits property of the derived class.
- Public - The inheritance is imported as Rational Rhapsody public inheritance.

Default = Property

VirtualInheritanceAsNon

The VirtualInheritanceAsNon property specifies whether virtual inheritance should be modeled as non-virtual.

- Property - The inheritance is imported to the VirtualInherits property of the derived class.
- Public - The inheritance is imported as Rational Rhapsody public inheritance.

Default = Property

Filtering

The Filtering metaclass contains properties that control which items are analyzed during the reverse engineering operation.

AnalyzeGlobalFunctions

The AnalyzeGlobalFunctions property specifies whether to analyze global functions. (Default = True)

AnalyzeGlobalTypes

The AnalyzeGlobalTypes property specifies whether to analyze global types. (Default = True)

AnalyzeGlobalVariables

The AnalyzeGlobalVariables property specifies whether to analyze global variables. (Default = True)

CreateReferenceClasses

The CreateReferenceClasses property specifies whether to create external classes for undefined classes that result from forward declarations and inheritance. By default, reference classes are created. If the incomplete class cannot be resolved, the tool deletes the incomplete class if this property is set to False. In some cases, the class cannot be deleted (for example, a class referenced by a typedef type).

Default = True

IncludeInheritanceInReference

The IncludeInheritanceInReference property specifies whether to include inheritance information in reference classes. (Default = False)

ReferenceClasses

The ReferenceClasses property specifies which classes to model as reference classes. Reference classes are classes that can be mentioned in the final design as placeholders without having to specify their internal details. For example, you can include the MFC classes as reference classes, without having to specify any of their members or relations. They would simply be modeled as terminals for context, to show that they are acting as superclasses or peers to other classes.

Default = empty string

ReferenceDirectories

The ReferenceDirectories property specifies which directories (and subdirectories) contain reference classes.

Default = empty string

ImplementationTrait

The ImplementationTrait metaclass contains properties that determine the implementation traits used during the reverse engineering operation.

AnalyzeIncludeFiles

The AnalyzeIncludeFiles property specifies which, if any, include files should be analyzed during reverse engineering. The possible values are as follows:

- AllIncludes - Analyze all include files.
- IgnoreIncludes - Ignore all include files.
- OnlyFromSelected - Analyze the specified include files only.
- OnlyLogicalHeader - Analyze the logical header files only.

Default = OnlyFromSelected

CreateDependencies

The CreateDependencies property is used during reverse engineering (RE) for creating dependencies from include statements found in the imported code. This property determines whether the RE utility creates dependencies. Reverse engineering imports include statements as dependencies if the option Create Dependencies from Includes is set in the Rational Rhapsody GUI. This operation is successful if the reverse engineering utility analyzes both the included file and the source - and the source and included files contain class declarations for creating the dependencies between them. If there is not enough information, the includes are not converted dependencies. This can happen in the following cases:

- The include file was not found, or is not in the scope Input tab settings.
- A class is not defined in the include file or source file, so the dependency could not be created.

If the dependency is not created successfully, the include files that were not converted to dependencies are imported to the Ada_CG::Class::SpecIncludes or ImpIncludes properties so you do not have to re-create them manually. If the include file is in the specification file, the information is imported to the SpecIncludes property; if it is in the implementation file, the information is imported to the ImpIncludes property. If a file contains several classes, include information is imported for all the classes in the file. The possible values for this property are as follows:

- None - Nothing is imported from include statements.

- DependenciesOnly - Model dependencies are created from include statements when it is possible to do so.
- All - The reverse engineering utility attempts to map the include file as a dependency. If it fails, the information is written to a property.

In addition to influencing reverse engineering, the CreateDependencies property also impacts the reverse engineering of user code added to model elements. The rules for interpreting #include and friend declarations for reverse engineering are as follows:

- Any #include OTHER in FILE is represented as a Uses dependency between each (outer) packages or classes in FILE to any (outer) packages or class in OTHER.
- If OTHER is not a specification file, the information is lost.
- If FILE is a specification file, the RefereeEffect is Specification. If FILE is an implementation file, the RefereeEffect is Implementation. Otherwise, the information is lost.

The way to decide if a file is a specification or an implementation file is defined elsewhere. 2. Any forward of a class or a package (via namespace) E in FILE is represented as a Uses dependency between each (outer) packages/classes in FILE to E. The RefereeEffect is Existence 3. This dependency is not added, if a Uses dependency can be matched. 4. Redundant Uses dependencies are removed. For example, when a relation is synthesized from a pointer to B, it is not necessary to add a Uses dependency. 5. A friend F (only when F is a class) of class C is is represented as a dependency with DependencyType to be Friendship from F to C.

Default = All

CreateFilesIn

The CreateFilesIn property is a placeholder for the reverse engineering option Create File-s In option. You should not set this value directly. The default value for C is Package; the default values for the other languages is None.

DataTypesLibrary

The DataTypesLibrary specifies type libraries, such as MFC, which contain predefined data types to which you can map imported classes. The default value for C and Java is an empty string; the default value for C++ is MFC.

ImportAsExternal

The ImportAsExternal property is a placeholder for the reverse engineering option Import as External. See the Rational Rhapsody Help for more information on this option. You should not change the value of this property directly.

Default = False

ImportDefineAsType

The ImportDefineAsType property is a Boolean value that specifies how to import a #define. The possible

values are as follows:

- True - Import a #define as a user type.
- False - Import a #define as a constant variable, constant function, or type according to the following policy:
- If the #define has parameters, Rational Rhapsody creates a constant function. This applies to Rational Rhapsody Developer for C only.
- If the #define does not have parameters and its value includes only one line, Rational Rhapsody creates a constant variable. In Rational Rhapsody Developer for C++, the property CG::Attribute::ConstantVariableAsDefine is set to True.
- If the #define was not imported as a variable or function, Rational Rhapsody creates a type.

Default = False

ImportStructAsClass

The ImportStructAsClass property is a Boolean value specifies how structs in external code are imported during reverse engineering. The possible values are as follows:

- True - structs are imported as classes.
- False - structs are imported as types of kind Structure.

Default = False

MapToPackage

The MapToPackage property specifies how code constructs are mapped to packages. The possible values are as follows:

- User - You specify the package to which all code constructs are mapped.
- Directory - Each directory is mapped to a separate package.FromExisting - Each package is mapped from an existing package. Note that this value is used only by the roundtrip and cannot be used directly. It is not an option for the reverse engineering GUI in Rational Rhapsody.

The default for C and C++ is User; the default for Java is Directory. (Default = Directory)

PreCommentSensibility

The PreCommentSensibility property specifies the difference in line numbers between an element and its comment in the code. A value of 0 means that the element comment should be placed on the same line as the element itself. If the difference in line numbers is more than the value set by this property, the comment is not imported as the element's description. (Default = 2)

ReflectDataMembers

The ReflectDataMembers property specifies whether to set the visibility of attributes in the features window from the access level of data members in the legacy code, and set the Visibility property to the

fromAttribute value. In previous versions of Rational Rhapsody, the RE utility imported all code data members as attributes with public visibility, and imported the access level of data members in the code into the Visibility property of the attributes. Beginning with Version 4.1, you can see the correct visibility of attributes in diagrams, and code generation will set the correct access level for generated attributes. In addition, this new functionality prevents the generation of redundant helper operations and generation of two sets of getter/setter functions (helpers) for imported attributes by disabling code generation for helpers. The possible values for the ReflectDataMembers property are as follows:

- None - The access level of data members is imported to the Visibility property for attributes and the DataMemberVisibility for relations. The visibility in the browser is always public.
- This is the behavior of previous versions of Rational Rhapsody.
- VisibilityOnly - The access level of data members is imported to the visibility of attributes in the browser. The Visibility property is always fromAttribute; for relations, the access level is imported to the ataMemberVisibility property.
- VisibilityAndHelpers - The access level of data members is displayed in the browser. The Visibility property is always set to fromAttribute (as VisibilityOnly). For relations, the access level is imported to the DataMemberVisibility property. In addition, generation of helper functions is disabled on the class properties level.
- The following table lists the property values that is set if you set ReflectDataMembers to VisibilityAndHelpers.

Subject and Metaclass Property Value CG::Relation AddComponentHelpersGenerate Cleared AddGenerate Cleared AddHelpersGenerate Cleared ClearGenerate Cleared ClearHelpersGenerate Cleared CreateComponentGenerate Cleared CG::Class InitCleanUpRelations Cleared

Default = VisibilityAndHelpers

UserDataTypes

The UserDataTypes specifies classes to be modeled as data types. This property corresponds to types entered in the Add Type window.

Default = empty string

UserPackage

The UserPackage property specifies the name of the package to which all code constructs are imported if MapToPackage is set to User.

Default = ReverseEngineering

Main

The Main metaclass contains properties that control the extensions used for implementation and specification files.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat , working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is as follows: ToTalNumberOfTokens=2,FileTokenPosition=1, LineTokenPosition=2

ImplementationExtension

The ImplementationExtension property specifies the file extensions used to filter the list of files displayed in the Add Files window of the reverse engineering tool. The default values are language-dependent:

- C - .c
- C++ - c,cpp,cxx,cc
- Java - Empty string

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. The default value is as follows: "([a-zA-Z_]+[:0-9a-zA-Z_.\/]*)"[:][ ]*LINE[ ]*([0-9]+)"

SpecificationExtension

The SpecificationExtension specifies the default extension used for specification files. The default values are language-dependent:

- C - h,inl
- C++ - h,hpp,hxx,inl
- Java - java

MFC

The MFC metaclass contains a property that affects the MFC type library.

DataTypes

The DataTypes property specifies classes to be modeled as MFC data types. There is only one predefined library (MFC) that contains only one class (Cstring). You can, however, expand this short list of classes by the addition of classes in this property or the creation of new libraries in the property files factory.prpfactory and site.prpsite.

Default = Cstring

MSVC60

The MSVC60 metaclass contains properties used to control the Microsoft Visual C++ environment.

Defined

The Defined property specifies symbols that are defined for the Microsoft Visual C++ version 6.0 (MSVC60) preprocessor. These symbols are automatically filled into the Name list of the Preprocessing tab of the Reverse Engineering Options window when you select Add > Dialect: MSVC60. The default value is as follows:

__STDC__,__STDC_VERSION__,__cplusplus,__DATE__,
__TIME__,_WIN32,_cdecl,__cdecl,__int64=int,__stdcall,
__export,_export,_AFX_PORTABLE,_M_IX86=500,__declspec,
__MSC_VER=1200,__inline=inline,__far,__near,_far,_near,
__pascal,_pascal,__asm,__finally=catch,__based,
__inline=inline,__single_inheritance,__cdecl,__int8=int,
__stdcall,__declspec,__int16=int,__int32=int,__try=try,
__int64=int,__virtual_inheritance,__except=catch, __leave=catch,__fastcall,__multiple_inheritance)

IncludePath

The IncludePath property specifies necessary include paths for the Microsoft Visual C++ preprocessor. It is possible to specify the path to the site installation of the compiler as part of the site.prp, thus doing it only once and not for every project.

Default = empty string

Undefined

The Undefined property specifies symbols that must be undefined for the Microsoft Visual C++ preprocessor. (Default = empty string)

Parser

The Parser metaclass contains properties that control symbols and macros used during reverse engineering.

Defined

The Defined property specifies symbols and macros to be defined using #define. For example, you can enter the following to define name as text with the appropriate intermediate character: /D name{=|#}text

Default = empty string

Dialects

The Dialects property specifies which symbols are added to the Preprocessing tab of the Reverse Engineering dialog box when that dialect is selected. The default value is MSVC60, which is itself defined by a metaclass of the same name under subject Ada_ReverseEngineering. This dialect specifies the symbols that are defined for the Microsoft Visual C++ environment. You can define your own dialect (in the site.prp file) and select it in the Dialects property. The default value for C is an empty string; the default value for C++ is MSVC60.

IncludePath

The IncludePath property enables you to specify an include path for the parser.

Default = empty string

Undefined

The Undefined property specifies symbols and macros to be undefined using #undef.

Default = empty string

Promotions

The Promotions metaclass contains a property that determines whether promotions are enabled during reverse engineering.

EnableAttributeToRelation

The EnableAttributeToRelation property is a Boolean value that specifies whether the “attribute to relation” promotion is enabled. If this is True, a data member found in the code should be represented as a relation in the model.

Default = True

EnableFunctionToObjectBasedOperation

The EnableFunctionToObjectBasedOperation property specifies whether object-based promotion is enabled during reverse engineering. Object-based promotion “promotes” a global function to comply with the pattern specified in the properties C_CG::Operation::PublicName and ProtectedName to be an operation of the class (object_type) defined in the function’s me parameter.

Default = False

EnableResolveIncompleteClasses

The EnableResolveIncompleteClasses property is a Boolean value that specifies whether the “resolve incomplete class” promotion is enabled. Incomplete promotions are the connects not found by the reverse engineering classes with either the classes in the model, or the classes that were imported during another reverse engineering session. The default value for C++ is True; the default value for Java is False.

EnableResolveIncompleteClass

The EnableResolveIncompleteClass property is a Boolean value that specifies whether the “resolve incomplete class” promotion is enabled. If this is True, the Reverse Engineering operation should find a class in the model on meeting a forward declaration in the code. (Default = True) NOT in C++; not in JAVA

ADA_Roundtrip

The ADA_Roundtrip subject contains properties that affect roundtripping.

General

The General metaclass contains properties that control how changes to code are roundtripped in Rational Rhapsody.

NotifyOnInvalidatedModel

The NotifyOnInvalidatedModel property is a Boolean value that determines whether a warning window is displayed during roundtrip. This warning is displayed when information might get lost because the model was changed between the last code generation and the roundtrip operation. (Default = True)

ParserErrors

The ParserErrors property specifies the behavior of roundtrip when a parser error is encountered. The possible values are as follows:

- Abort - Abort roundtrip whenever there is a parser error in the code. No changes are applied to the model.
- AskUser - When Rational Rhapsody encounters an error, it asks what you want to do. This option is available in Rational Rhapsody Developer for C++ only.
- AbortOnCritical - Abort roundtrip if any critical parser errors are encountered in the code.
- Ignore - Continue roundtrip, ignoring any parser errors that are encountered.

Default = Abort

PredefineIncludes

The PredefineIncludes property specifies the predefined include path for roundtripping. The default value for C++ is an empty string. The default value for Java is as follows:
$OMROOT\LangJava\src,D:\jdk1.2.2\src

PredefineMacros

The PredefineMacros property specifies the predefined macros for roundtripping. The default value is as follows:

DECLARE_META(class_0\,animClass_0), DECLARE_REACTIVE_META(class_0\,animClass_0), OMINIT_SUPERCLASS(class_0Super\,animClass_0Super), OMREGISTER_CLASS\,DECLARE_META_T(class_0\, ttype\,animClass_0),

DECLARE_REACTIVE_META_T(class_0\, ttype\,animClass_0),
DECLARE_META_SUBCLASS_T(class_0\, ttype\,animClass_0),
DECLARE_REACTIVE_META_SUBCLASS_T(class_0\, ttype\,animClass_0),
DECLARE_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM),
IMPLEMENT_META(class_0\,Default\,FALSE),
IMPLEMENT_META_S(class_0\,FALSE\,class_1\,animClass_1\, animClass_0),
IMPLEMENT_META_M(class_0\, FALSE\, class_0Super\, 2\,animClass_0),
IMPLEMENT_REACTIVE_META(class_0\,Default\,FALSE),
IMPLEMENT_REACTIVE_META_S(class_0\,FALSE\,class_1\, animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_REACTIVE_META_SIMPLE(class_0\,Default\,FALSE),
IMPLEMENT_REACTIVE_META_S_SIMPLE(class_0\,FALSE\,class_1\ ,animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M_SIMPLE(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_META_T(class_0\, Default\, FALSE\, animClass_0),
IMPLEMENT_META_S_T(class_0\,FALSE\,class_0Super\,animclas s_0Super\,animClass_0),
IMPLEMENT_META_M_T(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_META_OBJECT(class_0\,class_type\,Default\, FALSE),
IMPLEMENT_META_S_OBJECT(class_0\,class_type\,FALSE\, class_1\,animClass_1\,animClass_0),
IMPLEMENT_META_M_OBJECT(class_0\,class_type\,FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_REACTIVE_META_OBJECT(class_0\,class_type\, Default\,FALSE),
IMPLEMENT_REACTIVE_META_S_OBJECT(class_0\,class_type\,
FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M_OBJECT(class_0\,class_type\, FALSE\, class_0Super\, 2
\,animClass_0), IMPLEMENT_REACTIVE_META_SIMPLE_OBJECT(class_0\,
class_type\,Default\,FALSE), IMPLEMENT_REACTIVE_META_S_SIMPLE_OBJECT(class_0\,
class_type\,FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M_SIMPLE_OBJECT(class_0\, class_type\,FALSE\, class_0Super\,
2 \,animClass_0), IMPLEMENT_META_T_OBJECT(class_0\,class_type\, Default\, FALSE\,
animClass_0), IMPLEMENT_META_S_T_OBJECT(class_0\,class_type\,FALSE\,
class_0Super\,animclass_0Super\,animClass_0),
IMPLEMENT_META_M_T_OBJECT(class_0\,class_type\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM\,
INCREMENTNUM\,ISPROTECTED), DECLARE_META_PACKAGE(Default),
DECLARE_PACKAGE(Default), IMPLEMENT_META_PACKAGE(Default\,Default),
DECLARE_META_EVENT(event_0), DECLARE_META_SUBEVENT(event_0\,event_0Super\,
event_0SuperNamespace), IMPLEMENT_META_EVENT(event_0\,Default\,event_0),
IMPLEMENT_META_EVENT_S(words\, words\, baseWords),
DECLARE_OPERATION_CLASS(mangledName), DECLARE_META_OP(mangledName),
OM_OP_UNSER(type\, name), OP_UNSER(func\, name), OP_SET_RET_VAL(retVal),
OM_OP_SET_RET_VAL(retVal), IMPLEMENT_META_OP(animatedClassName\, mangledName\,
opNameStr\, isStatic\, signatureStr\, numOfArgs), IMPLEMENT_OP_CALL(mangledName\,
userClassName\, call\, retExp), STATIC_IMPLEMENT_OP_CALL(mangledName\, userClassName\,
call\, retExp), OMDefaultThread=0, NULL=0, OMDECLARE_GUARDED
OM_DECLARE_COMPOSITE_OFFSET

ReportChanges

The ReportChanges property defines which changes are reported (and displayed) by the roundtrip operation. The possible values are as follows:

- None—No changes are displayed in the output window.
- AddRemove—Only the elements added to, or removed from, the model are displayed in the output window.

- UpdateFailures—Only unsuccessful changes to the model are displayed in the output window.
- All—All changes to the model are displayed in the output window.

(Default = AddRemove)

RestrictedMode

The RestrictedMode property is a Boolean value that specifies whether restricted-mode roundtripping is enabled. This property can be modified on the configuration level. Restricted mode of full roundtrip enables you to roundtrip unusual usage of Rational Rhapsody elements, such as a class declaration in a user-defined type. Restricted mode has more limitations, but preserves the model from unexpected changes. The additional limitations for restricted mode are as follows:

- User-defined types cannot be removed or changed on roundtrip because Rational Rhapsody code generation adds the Ignore annotation for a user-defined type declaration.
- Relations cannot be removed or changed on roundtrip.
- New classes are not added to the model.

(Default = False)

RoundtripScheme

The RoundtripScheme property specifies whether to perform a basic or full roundtrip. Batch and online roundtrips change their behavior according to the specified value.

Default = Basic

CPP_Roundtrip::Type

The Type metaclass contains a property that controls whether user-defined types are ignored during the roundtrip operation.

Ignore

The Ignore property is a Boolean value that specifies whether to include user-defined types in a roundtrip operation. Types with the Ignore property set to True are generated with an Ignore annotation and will not be changed when a roundtrip is performed. The default value of this property is True, which allows no deletion or change to be done on types. Setting this property to False will reflect changes to the types declaration and deletion of types during roundtrip. Modifying the name of an existing type results in the addition of a new type, and removal of the model type (if the AcceptChanges property allows element removal), and the model's references to the removed type is lost (such as appearance in diagrams, property settings, and so on). You can set this property either on the configuration or on specific elements in the model (which will affect itself and its aggregates). (Default = True)

Update

The Update metaclass contains a property that controls the update process used during roundtripping.

AcceptChanges

The AcceptChanges property is an enumerated type that specifies which changes are applied to each CG element (attribute, operation, type, class, or package). You can apply separate properties to each type of CG element. The possible values are as follows:

- All—All the changes can be applied to the model element.
- NoDelete—All the changes except deletion can be applied to the model element. This setting prevents accidental removal of operations, constructors, attributes, relations, variables, instances, and functions.
- AddOnly—Apply only the addition of an aggregate to the model element. You cannot delete or change elements.
- NoChanges—Do not apply any changes to the model element.

Note that the value of the property is propagated to all the aggregates of an element. Therefore, if a package has the property value NoChanges, no elements in that package is changed.

(Default = NoDelete)

Animation

The Animation subject contains properties that support black box animation. It contains a single metaclass: ClassifierRole.

ClassifierRole

The ClassifierRole metaclass contains properties that control black box animation.

DisplayMessagesToSelf

The DisplayMessagesToSelf property determines whether messages-to-self are displayed during animation. The possible values are as follows:

- None - Do not display any messages-to-self.
- All - Display all messages-to-self.

For example, if lifeline L1 is mapped to objects O1 and O2, you can suppress the messages between O1 and O2 for this lifeline. Even if a lifeline is mapped to a single object, the messages the object sends to itself (for example, timeout events) can be suppressed. (Default = All)

MappingPolicy

The MappingPolicy property specifies how lifelines are mapped during animation. You can set these properties for every lifeline in the diagram. The possible values are as follows:

- Smart - Rational Rhapsody decides the mapping policy.
- If the lifeline has a reference sequence diagram, the mapping is equivalent to ObjectAndDerivedFromRefSD; otherwise, the mapping is equivalent to ObjectAndItsParts (which, for an object without any parts, is the same as SingleObject).
- ObjectAndItsParts - The lifeline (classifier role) is mapped to the object that matches the name of the lifeline and all its parts (recursively), excluding parts that are explicitly shown in the diagram.
- This is the default value for a lifeline realized by a composite class that does not reference sequence diagrams (SDs) when Smart mode is used.
- SingleObject - The lifeline is mapped to a single, run-time object.
- ObjectAndDerivedFromRefSD - The lifeline is mapped to an object by its role name (if it exists) and to all derived objects from the reference SDs (according to their mapping rules).
- This is the default value for lifelines decomposed by reference SDs when Smart mode is used.

Note the following:

- If the lifeline can be mapped to a composite object, the compositional hierarchy is ignored in this mapping.
- Parts that are represented by other lifelines are excluded. For example, if two lifelines are decomposed

to the same reference SD, one of them is completely ignored (arbitrarily).

(Default = SingleObject)

ATL

The ATL subject contains the following metaclasses:

- Class
- Configuration
- Macro
- Operation

Class

The Class metaclass contains properties that control the behavior of ATL classes.

Aggregation

The Aggregation property enables or disables aggregation for a <<COM ATL Class>>. Note that this property must be set for the aggregated (part) class, not the aggregate (whole) class.

Default = Yes

ConnectionPoints

The ConnectionPoints property specifies whether an ATL class supports a connection point.

Default = No

DeclarationModifier

The DeclarationModifier property is a class modifier for an ATL class that gets printed before the class name.

Default = ATL_NO_VTABLE

DeclareClassFactory

The DeclareClassFactory property specifies a template for the generation of code for ATL classes. The following is an example of a template that could be specified for this property:

class $DeclarationModifier $class : public CComObjectRootEx$ThreadModel, public CCOMCoClass$class, CLSID_def, public IDispatchImplIdef, IID_Idef, LIBID_ALL_KIND_OF_ATLLib { ... }

This template references information stored in the ATLRootClass - the ATLClassObject and the IDispatchImpl properties.

Default = Empty string

FreeThreadedMarshaller

The FreeThreadedMarshaller property specifies whether an ATL class supports a free-threaded marshaller.

Default = No

SupportErrorInfo

The SupportErrorInfo property specifies whether an ATL class supports the ISupportErrorInfo interface.

Default = No

ThreadingModel

The ThreadingModel property specifies the mapping of Rational Rhapsody threads (and any other UML modeling construct) and the COM apartment/threading model for the class under design. Because there is no direct mapping, you can freely define the apartment model for every generated ATL class.

Default = Apartment

Configuration

The Configuration metaclass contains properties that control the configuration of ATL classes.

APPID

The APPID property specifies the APPID name. If you do not specify the APPID property at the ComponentConfiguration level, Rational Rhapsody generates it automatically.

Default = Empty string

ATLCustomCPFireOperation

The ATLCustomCPFireOperation property is a template for the implementation of connection point fire operations for custom interfaces.

Default = $opRetType Fire_$opname($arguments) { $opRetType ret; T pT = static_cast<T*>(this); int*

nConnectionIndex; int nConnections = m_vec.GetSize(); for (nConnectionIndex = 0; nConnectionIndex < nConnections; nConnectionIndex++) { pT->Lock(); CComPtr<IUnknown> sp = m_vec.GetAt(nConnectionIndex); pT->Unlock(); $interface p$interface = reinterpret_cast<$interface*>(sp.p); if (p$interface != NULL) { ret = p$interface->$opname($argumentlist); } } return ret; }*

ATLDispInterfaceCPFireOperation

The ATLDispInterfaceCPFireOperation property is a template for the implementation of connection point fire operations for dual interfaces.

Default = $opRetType Fire_$opname($arguments) { CComVariant varResult; T pT = static_cast<T*>(this); int nConnectionIndex; CComVariant* pvars = NULL ; int noOfArgs = $noOfArgs ; if(noOfArgs > 0) { pvars = new CComVariant[noOfArgs]; } int nConnections = m_vec.GetSize(); for (nConnectionIndex = 0; nConnectionIndex < nConnections; nConnectionIndex++) { pT->Lock(); CComPtr<IUnknown> sp = m_vec.GetAt(nConnectionIndex); pT->Unlock(); IDispatch* pDispatch = reinterpret_cast<IDispatch*>(sp.p); if (pDispatch != NULL) { VariantClear(&varResult); /*Add your code to initilize pvars[..]*/ DISPPARAMS disp = { pvars, NULL, $noOfArgs, 0 }; pDispatch->Invoke($id, IID_NULL, LOCALE_USER_DEFAULT, DISPATCH_METHOD, &disp, &varResult, NULL, NULL); } } delete[] pvars; return varResult.scode; }*

ATLProxyClass

The ATLProxyClass property provides a template for generating the ATL proxy class for a connection point.

Default = #ifndef CP$interface_H #define CP$interface_H $import template <class T> class CProxy$interface : public IConnectionPointImpl<T, &$IDinterface, CComDynamicUnkArray> { public: $operations }; #endif

CPP_StandardInclude

The CPP_StandardInclude property adds standard include files in all CPP files.

Default = stdafx.h

InProcServerExports

The InProcServerExports property provides a template for the DEF file used during DLL creation.

Default =

EXPORTS DllCanUnloadNow @1 PRIVATE DllGetClassObject @2 PRIVATE DllRegisterServer @3 PRIVATE DllUnregisterServer @4 PRIVATE

InProcServerMainLineTemplate

The InProcServerMainLineTemplate property provides a template for the Dllmain() function.

Default =

if (dwReason == DLL_PROCESS_ATTACH) { _Module.Init(ObjectMap, hInstance/*, &LIBID_$PackageLib*/); DisableThreadLibraryCalls(hInstance); } else if (dwReason == DLL_PROCESS_DETACH) { _Module.Term(); } return TRUE; // ok

InProcServerMainModule

The InProcServerMainModule property provides a template for the declaration and definition of each of the COM methods exported in the DLL. The default exported methods are DllCanUnloadNow(), DllGetClassObject(), llRegisterServer(), and DllUnregisterServer().

Default =

CComModule _Module; #ifdef _ATL_STATIC_REGISTRY #include <statreg.h> #if (_MSC_VER < 1310) // Avoid in .NET 2003 #include <statreg.cpp> #endif #endif #if (_MSC_VER < 1310) // Avoid in .NET 2003 #include <atlimpl.cpp> #endif static BOOL aFlag = TRUE ; // // Used to determine whether the DLL can be unloaded by OLE STDAPI DllCanUnloadNow(void) { if(_Module.GetLockCount()==0) { OXF::end(); aFlag = TRUE ; return S_OK ; } else return S_FALSE ; } // // Returns a class factory to create an object of the requested type STDAPI DllGetClassObject(REFCLSID rclsid, REFIID riid, LPVOID* ppv) { if(aFlag) { if(OXF::init(0, NULL, 6423)) OXF::start(TRUE); aFlag = FALSE ; } return _Module.GetClassObject(rclsid, riid, ppv); } // // DllRegisterServer - Adds entries to the system registry STDAPI DllRegisterServer(void) { // registers object, typelib and all interfaces in typelib RegisterApp(COMPAPPID, "$component") ; return _Module.RegisterServer($RegTlb); } // // DllUnregisterServer - Removes entries from the system registry STDAPI DllUnregisterServer(void) { return _Module.UnregisterServer($RegTlb); }

InProcStdAfx

The InProcStdAfx property is a template for the COM InProcServer StdAfx.h header file.

Default =

#if _MSC_VER > 1000 #pragma once #endif // _MSC_VER > 1000 #define STRICT #ifndef _WIN32_WINNT #define _WIN32_WINNT 0x0400 #endif #define _ATL_APARTMENT_THREADED #include <atlbase.h> //You may derive a class from CComModule and use it if you want to override //something, but do not change the name of _Module extern CComModule _Module; #include <atlcom.h> #include "RhapRegistery.h"

OutProcServerMainLineTemplate

The OutProcServerMainLineTemplate property provides a template for the Dllmain() function.

Default =

_Module.StartMonitor(); #if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED) hRes = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE | REGCLS_SUSPENDED); _ASSERTE(SUCCEEDED(hRes)); hRes = CoResumeClassObjects(); #else hRes = _Module.RegisterClassObjects(CLSCTX_LOCAL_SERVER, REGCLS_MULTIPLEUSE); #endif _ASSERTE(SUCCEEDED(hRes)); MSG msg; while (GetMessage(&msg, 0, 0, 0)) DispatchMessage(&msg); _Module.RevokeClassObjects(); Sleep(dwPause); //wait for any threads to finish _Module.Term(); CoUninitialize();

OutProcServerMainModule

The OutProcServerMainModule property provides a template for the declaration and definition of each of the COM methods exported from the DLL. The default exported methods are DllCanUnloadNow(), DllGetClassObject(), llRegisterServer(), and DllUnregisterServer().

Default =

#ifdef _ATL_STATIC_REGISTRY #include <statreg.h> #if (_MSC_VER < 1310) // Avoid in .NET 2003 #include <statreg.cpp> #endif #endif #if (_MSC_VER < 1310) // Avoid in .NET 2003 #include <atlimpl.cpp> #endif const DWORD dwTimeOut = 5000; // time for EXE to be idle before shutting down const DWORD dwPause = 1000; // time to wait for threads to finish up // Passed to CreateThread to monitor the shutdown event static DWORD WINAPI MonitorProc(void* pv) { CExeModule* p = (CExeModule*)pv; p->MonitorShutdown(); return 0; } LONG CExeModule::Unlock() { LONG l = CComModule::Unlock(); if (l == 0) { bActivity = true; SetEvent(hEventShutdown); // tell monitor that we transitioned to zero } return l; } //Monitors the shutdown event void CExeModule::MonitorShutdown() { while (1) { WaitForSingleObject(hEventShutdown, INFINITE); DWORD dwWait=0; do { bActivity = false; dwWait = WaitForSingleObject(hEventShutdown, dwTimeOut); } while (dwWait == WAIT_OBJECT_0); // timed out if (!bActivity && m_nLockCnt == 0) { // if no activity, shut down #if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED) CoSuspendClassObjects(); if (!bActivity && m_nLockCnt == 0) #endif break; } } CloseHandle(hEventShutdown); PostThreadMessage(dwThreadID, WM_QUIT, 0, 0); } bool CExeModule::StartMonitor() { hEventShutdown = CreateEvent(NULL, false, false, NULL); if (hEventShutdown == NULL) { return false; } DWORD dwThreadID; HANDLE h = CreateThread(NULL, 0, MonitorProc, this, 0, &dwThreadID); return (h != NULL); } CExeModule _Module; LPCTSTR FindOneOf(LPCTSTR p1, LPCTSTR p2) { while (p1 != NULL && *p1 != NULL) { LPCTSTR p = p2; while (p != NULL && *p != NULL) { if (*p1 == *p) { return CharNext(p1); } p = CharNext(p); } p1 = CharNext(p1); } return NULL; }

OutProcServerRegistration

The OutProcServerRegistration property provides a registration template for a COM OutProcServer server. This is inserted into the OutProcServer main file (in the WinMain(...) function) for server registration.

Default =

lpCmdLine = GetCommandLine(); //this line necessary for _ATL_MIN_CRT #if _WIN32_WINNT >= 0x0400 & defined(_ATL_FREE_THREADED) HRESULT hRes = CoInitializeEx(NULL, COINIT_MULTITHREADED); #else HRESULT hRes = CoInitialize(NULL); #endif _ASSERTE(SUCCEEDED(hRes)); _Module.Init(ObjectMap, hInstance); /*, &LIBID_$package);*/ _Module.dwThreadID = GetCurrentThreadId(); TCHAR szTokens[] = _T("-/"); int nRet = 0; BOOL bRun = TRUE; LPCTSTR lpszToken = FindOneOf(lpCmdLine, szTokens); while (lpszToken != NULL) { if

(lstrcmpi(lpszToken, _T("UnregServer"))==0) { nRet = _Module.UnregisterServer($RegTlb); bRun = FALSE; break; } if (lstrcmpi(lpszToken, _T("RegServer"))==0) { nRet = _Module.RegisterServer($RegTlb); RegisterApp(COMPAPPID, "$component") ; bRun = FALSE; break; } lpszToken = FindOneOf(lpszToken, szTokens); } if(!bRun) { _Module.Term(); CoUninitialize(); return nRet; }

OutProcStdAfx

The OutProcStdAfx property is a template for COM OutProcServer StdAfx.h header file.

Default =

#if _MSC_VER > 1000 #pragma once #endif // _MSC_VER > 1000 #define STRICT #ifndef _WIN32_WINNT #define _WIN32_WINNT 0x0400 #endif #define _ATL_APARTMENT_THREADED #include <atlbase.h> //You may derive a class from CComModule and use it if you want to override //something, but do not change the name of _Module class CExeModule : public CComModule { public: LONG Unlock(); DWORD dwThreadID; HANDLE hEventShutdown; void MonitorShutdown(); bool StartMonitor(); bool bActivity; }; extern CExeModule _Module; #include <atlcom.h> #include "RhapRegistery.h"

ProxyStubExports

The ProxyStubExports property specifies the content of the ProxyStub.dll file. You can modify this content as desired.

Default =

EXPORTS DllGetClassObject @1 PRIVATE DllCanUnloadNow @2 PRIVATE DllRegisterServer @3 PRIVATE DllUnregisterServer @4 PRIVATE

RegistrationModule

The RegistrationModule property specifies the name of the file that implements the ATL class registration.

Default = RhapRegistery

ServerNonCreatableObjectMapEntry

The ServerNonCreatableObjectMapEntry property is a macro that specifies a non-creatable ALT class entry into the ALT server map.

Default =

OBJECT_ENTRY_NON_CREATEABLE($class)

The $class keyword is replaced with the name of the ATL class.

ServerObjectMapBegin

The ServerObjectMapBegin property sets the ATL server map macro to "begin."

Default = BEGIN_OBJECT_MAP(ObjectMap)

ServerObjectMapEnd

The ServerObjectMapEnd property sets the ATL server map macro to "end."

Default = END_OBJECT_MAP()

ServerObjectMapEntry

The ServerObjectMapEntry property is a template that specifies a creatable ATL class entry into the ATL server map.

Default = OBJECT_ENTRY(CLSID_$coclass, $class)

The $coclass keyword is replaced with the name of the coclass; $class is replaced with the name of the ATL class that implements the coclass.

TypeLibImportFormat

The TypeLibImportFormat specifies the template used to generate COM TLB import statements.

Default =

#import "$tlbPath" raw_interfaces_only, raw_native_types, no_namespace, named_guids

Macro

The Macro metaclass contains properties that act as templates for ATL classes and operations.

ATL_ErrorMethodBody

The ATL_ErrorMethodBody property is a template that implements the SupportErrorInfo operation.

Default =

static const IID* arr[] = { $IID_implClass }; for (int i=0; i < sizeof(arr) / sizeof(arr[0]); i++) { if (InlineIsEqualGUID(*arr[i],riid)) { return S_OK; } } return S_FALSE;

ATL_FTMCreateBody

The ATL_FTMCreateBody property is a template that causes a function in an ATL class to create a free-threaded marshaller.

Default =

return CoCreateFreeThreadedMarshaler(GetControllingUnknown(), &m_pUnkMarshaler.p);

ATL_FTMReleaseBody

The ATL_FTMReleaseBody property is a template that causes a function in an ATL class to release a free-threaded marshaller.

m_pUnkMarshaler.Release();

ATLClassObject

The ATLClassObject property specifies the ATL class that implements a COM coclass.

Default = CComCoClass<$class, &CLSID_$coclass>

The $class keyword is replaced with the name of the ATL class that implements the coclass; $coclass is replaced with the name of the coclass that exposes the COM interface.

ATLConnectionPointImpl

The ATLConnectionPointImpl property specifies the ATL class that implements the IConnectionPointContainer interface.

Default = IConnectionPointContainerImpl<$interface>

The $interface keyword is replaced with the name of the interface being implemented.

ATLRootClass

The ATLRootClass property specifies the ATL root class.

Default = CComObjectRootEx<$ThreadModel>

The $ThreadModel keyword is replaced with the value of the ThreadingModel property (Default = Apartment).

BeginConnectionPointMap

The BeginConnectionPointMap property specifies the macro to start a connection point map for an ATL class.

Default = BEGIN_CONNECTION_POINT_MAP($interface)

The $interface keyword is replaced with the interface that contains the connection point map.

BeginInterfaceMap

The BeginInterfaceMap property controls how macro templates are generated. The property specifies the start macro for a COM map of an ATL class.

Default = BEGIN_COM_MAP($class)

The $class keyword is replaced with the name of the ATL class.

ClassRegistration

The ClassRegistration property specifies the ATL class registration macro.

Default =

DECLARE_RHAPSODY_REGISTER(CLSID_$coclass, "$TypeName",
"$VersionIndepProgID","$ProgID", "$ThreadingModel", COMPAPPID)

The keywords are replaced with the appropriate information, as follows:

- $coclass - Replaced with the name of the coclass that the ATL class implements.
- $TypeName - Replaced with the value of the TypeName property, which specifies the declaration of the class type being registered (Default = $class).
- $VersionIndepProgID - Replaced with the value of the VersionIndepProgID property (Default = $component.$class).
- $ProgID - Replaced with the value of the ProgID property (Default = $component.$class.1).
- $ThreadModel - Replaced with the value of the ThreadingModel property (Default = Apartment).

ConnectionPointMapEntry

The ConnectionPointMapEntry property specifies the macro for the connection point map entry.

Default = CONNECTION_POINT_ENTRY($interface)

The $interface keyword is replaced with the interface that contains the connection point map.

ConnectionPointProxyClass

The ConnectionPointProxyClass property specifies the proxy class for the connection point.

Default = CProxy$interface< $class >

The $interface keyword is replaced with the name of the interface being implemented; $class is replaced with the ATL class.

DeclareControllingIUnknown

The DeclareControllingUnknown property prints the DECLARE_GET_CONTROLLING_UNKNOWN() macro into the ATL class. For more information on COM properties, see the MSDN Online Library.

Default = DECLARE_GET_CONTROLLING_UNKNOWN().

DeclareProtect

The DeclareProtect property specifies the macro that protects the ATL object from being deleted if, during FinalConstruct(), the nested object increments the reference count and then decrements the count to 0.

Default = DECLARE_PROTECT_FINAL_CONSTRUCT()

EndConnectionPointMap

The EndConnectionPointMap property specifies the macro to end a connection point map for an ATL class.

Default = END_CONNECTION_POINT_MAP()

EndInterfaceMap

The EndInterfaceMap property specifies the end macro for a COM map of an ATL class.

Default = END_COM_MAP()

IDispatchImpl

The IDispatchImpl property provides support for animation.

Default = IDispatchImpl<$interface,&IID_$interface, &LIBID_$Package>

The $interface keyword is replaced with the name of the interface being animated; $Package is replaced with the name of the COM library to which the interface belongs.

InterfaceEntry

The InterfaceEntry property specifies the ATL macro that defines the COM map interface entry point.

Default = COM_INTERFACE_ENTRY($interface)

The $interface keyword is replaced with the name of the interface being animated.

InterfaceEntry2

The InterfaceEntry2 property specifies the ATL macro that defines the COM map interface entry point, to disambiguate two branches of inheritance.

Default = COM_INTERFACE_ENTRY2($dupinterface,$interface)

The $dupinterface keyword is replaced with the name of the duplicate interface; $interface is replaced with the name of the interface being animated.

InterfaceEntryAggr

The InterfaceEntryAggr property is a macro that enables an interface entry of an aggregated object in an ALT class interface map.

Default = COM_INTERFACE_ENTRY_AGGREGATE(IID_$interface, $datamem)

The $interface keyword is replaced with the name of the interface being animated; $datamem is replaced with the data member.

NoAggregation

The NoAggregation property is a template for a macro that specifies that an object cannot be aggregated.

Default = DECLARE_NOT_AGGREGATABLE($class)

The $class keyword is replaced with the name of the ATL class.

OnlyAggregation

The OnlyAggregation property is a template for a macro that specifies that an object must be aggregated.

Default = DECLARE_ONLY_AGGREGATABLE($class)

The $class keyword is replaced with the name of the ATL class.

ProgID

The ProgID property is a standard MSDN COM property. This property returns the programmatic identifier (ProgID) for the specified OLE object. For more information on COM properties, see the MSDN Online Library (http://msdn.microsoft.com/library/).

Default = $component.$class.1

The $component keyword is replaced with the name of the component; $class is replaced with the name of the ATL class.

ReturnSuccess

The ReturnSuccess property is a macro that specifies a successful return of an ATL class standard operation.

Default = return S_OK;

SupportAggregation

The SupportAggregation property is a template for a macro, which specifies that an object can be aggregated.

Default = DECLARE_AGGREGATABLE($class)

The $class keyword is replaced with the name of the ATL class.

TypeName

The TypeName property specifies the declaration of the class type being registered, when you use the ClassRegistration property to specify the ATL class registration macro.

Default = $class class

The $class keyword is replaced with the name of the ATL class.

VersionIndepProgID

The VersionIndepProgID property specifies the version-independent ID when you use the ClassRegistration property to specify the ATL class registration macro.

Default = $component.$class

The $component keyword is replaced with the name of the component; $class is replaced with the name of the ATL class.

Operation

The Operation metaclass contains a property that specifies a standard ATL class operation.

STDMETHOD

The STDMETHOD property specifies a standard ATL class operation.

Default = Cleared

Browser

The Browser subject enables you to modify the display of the Rational Rhapsody browser.

Operation

The metaclass Operation contains properties that are used to control the way operations are displayed in the browser.

ShowReturnTypeFromCG

The property ShowReturnTypeFromCG is used to specify that the signatures displayed for operations in the browser should include the return type that is generated in the code.

This property does not affect the browser display if you are working in Label mode.

Default = Cleared

Settings

The Settings metaclass contains properties that control the display of the Rational Rhapsody browser.

DeleteConfirmation

The DeleteConfirmation property specifies whether confirmation is required before deleting a graphical element from the model. Note that this property does not apply to statechart elements, which have a separate DeleteConfirmation property. The possible values are as follows:

- Always - Rational Rhapsody displays a confirmation dialog each time you try to delete an item from the model.
- Never - Confirmation is not required to delete an element.
- WhenNeeded - Rational Rhapsody asks for confirmation if there are references to the element (or for some other reason).

Default = Always

DisplayMode

The DisplayMode property specifies the mode used to display the browser tree. The possible values are as follows:

- Meta-class - Display all the metaclass nodes (such as operations and objects).
- Flat - Do not display metaclass nodes.

Default = Meta-class

PreserveTreeNodeExpandState

PreserveTreeNodeExpandState is a Boolean property that allows you to specify whether or not Rational Rhapsody should remember the open/closed state of nested packages/folders in the browser for the duration of a Rational Rhapsody session.

To have Rational Rhapsody keep track of this information, set the value of the property to True.

Default = Checked

ShowAttributesCodeAsTooltip

When you hover over an attribute in the browser, a tooltip is displayed, showing the code that is generated for the attribute declaration. The boolean property ShowAttributesCodeAsTooltip allows you to turn this behavior off/on.

Default = Checked

ShowContextForAssociation

ShowContextForAssociation is a boolean property that allows you specify that for association ends, Rational Rhapsody should display the destination of the association in parentheses alongside the name of the association end in the browser.

This is particularly useful for situations where users may change the names of association ends, which by default refer to the destination (for example, itsClass_3).

Default = False

ShowFeatures

Reserved for future use.

Default = Checked

ShowImplementationArgument

The property ShowImplementationArgument controls the way that operation arguments are displayed in the browser.

Ordinarily, the browser displays just the argument type and name.

However, if you change the value of this property to True, the browser will show the exact code that is generated for the arguments, for example, "getData(const Vehicle& currentVehicle)" instead of "getData(Vehicle currentVehicle)".

Default = Cleared

ShowImplementationNameInTree

The ShowImplementationNameInTree property specifies whether to display the implementation (generated) name of operations instead of the design (user-assigned) name in the browser tree. The default is False (the design name is displayed). For example, in Rational Rhapsody Developer for C, if you create an operation named open() for an object named Valve, the operation's design name is open(), but its implementation name is actually Valve_open().You must always use the implementation name for operations (and states) anywhere you write code. You can toggle the display of operation names in the browser tree between implementation names and design names by changing this property and then closing and reopening the object node to refresh the display of the operation names. Note that the implementation name is always displayed in the Operation window on the right side of the browser, regardless of this property setting.

Default = Cleared

ShowLabels

The ShowLabels property is a Boolean value that specifies whether to display labels instead of names in the browser or diagrams, depending on which property is set.

Default = Cleared

ShowMultipleStereotypes

The property ShowMultipleStereotypes is a Boolean property in the browser. Setting this property to Cleared shows only the first stereotype of a certain element even if it has several stereotypes.

Default = Checked

ShowOrder

The boolean property ShowOrder enables/disables the ability to reorder elements in the browser by enabling/disabling the up/down arrow controls. When the user selects View > Browser Display Options > Enable Ordering from the main menu, the property is assigned the value Checked. When the user deselects the Enable Ordering menu item, the property is assigned the value Cleared.

ShowPredefinedPackage

The ShowPredefinedPackage property is a Boolean value that determines whether the PredefinedTypes package is displayed in the browser. When the property is set to Cleared, the package is hidden.

Default = Checked

ShowSourceArtifacts

When using reverse engineering and/or roundtripping, certain information necessary for the Rational Rhapsody code respect feature is stored as SourceArtifact elements.

By default, these SourceArtifact elements are not displayed in the browser.

If the property ShowSourceArtifacts is set to True, then these elements are opened in the browser.

This property corresponds to the menu option View > Browser Display Options > Show Source Artifacts.

Default = Cleared

ShowStereotypes

The property ShowStereotypes determines whether the browser displays the stereotype applied to a model element, alongside the name of the element. The possible values for this property are:

- No - stereotype is not displayed
- Prefix - stereotype is displayed to the left of the element name
- Suffix - stereotype is displayed to the right of the element name

Default = Prefix. The property is set at the project level. When the user selects View > Browser Display Options > Show Stereotype from the main menu, the property is assigned the value Prefix. When the user deselects the Show Stereotype menu item, the property is assigned the value No.

CG

The subject CG contains the following metaclasses for code generation properties that are common to all languages:

- Argument
- Attribute
- CGGeneral
- Class
- Component
- Configuration
- Dependency
- Event
- File
- General
- Generalization
- Operation
- Package
- Relation
- Statechart
- Type

Argument

The Argument metaclass contains a property that controls the animation of a specific argument.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation.

The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property check box is cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property check box is cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property check box is cleared, all the arguments are not animated.
- If the AnimateArguments property check box is cleared, all the arguments are not animated, regardless of the specific argument Animate property settings. (Default = Checked)

UsageType

The property UsageType determines how an #include is generated for a type used as an argument. The possible values are as follows:

- Existence - If the provider is a class, a forward class declaration is generated.
- Implementation - An #include statement is generated in the implementation file.
- Specification - An #include statement is generated in the specification file.
- None - no #include is generated. (Default = Implementation)

Attribute

The Attribute metaclass contains properties for implementing attributes and methods that handle attributes.

Accessor

The Accessor property specifies the format of the names of attribute accessors. The string get_$attribute means that if an accessor is generated for an attribute, it is called get_attributeName.

(Default = get$attribute:c)

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation.

The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property check box is cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property check box is cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property check box is cleared, all the arguments are not animated.
- If the AnimateArguments property check box is cleared, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

AnimateAttributes

The AnimateAttributes property specifies whether to animate class member attributes in the animated browser.

During instrumentation, Rational Rhapsody displays the values of class member attributes in the animated browser using the C++ stream output operator.

For this to work, the attribute must be of a built-in type that can be output to a stream, such as an int or a char*.

If, however, the attribute is of a complex, user-defined type that the compiler cannot serialize, trying to animate it will cause compilation errors.

In VxWorks, trying to animate attributes of type long unsigned int will also cause errors. If you want to animate attributes of user-defined types defined either inside or outside of Rational Rhapsody, you must do one of the following:

- Add to the framework an overloaded C++ stream output operator for the type, such as:

ostream operator(sometype);

- Instantiate the template string2X(T t) with the type.
- Disable the AnimateAttributes property by setting it to Cleared.

(Default = Checked)

CorbaRealizingAccessor

The CorbaRealizingAccessor property is a format string that specifies the naming convention of an attribute getter that realizes a CORBAInterface attribute. (Default = $attribute)

CorbaRealizingMutator

The CorbaRealizingMutator property is a format string that specifies the naming convention of an attribute setter that realizes a CORBAInterface attribute. (Default = $attribute)

Generate

The Generate property specifies whether to generate code for a particular type of element.

(Default = Checked)

Implementation

The Implementation property enables you to specify how Rational Rhapsody generates code for a given element (for example, as a simple array, collection, or list).

When this property is set to Default and the multiplicity is bounded (not *) and the type of the attribute is not a class, code is generated without using the container properties (as in previous versions of Rational Rhapsody).

Note that the software generates a single accessor and mutator for an attribute, as opposed to relations,

which can have several accessors and mutators.

In smart generation mode, a setter is not generated when the attribute is Constant and either:

- The attribute is not a Reference.
- or The multiplicity of the attribute is 1.
- or The CG::Attribute::Implementation property is set to EmbeddedScalar or EmbeddedFixed.

(Default = Default)

IsConst

IsConst refers to the getter of the attribute (it does not indicate whether or not the attribute itself is a constant).

This property can take one of the following values:

Signature - This value means that the getter function is constant. For an int attribute, the generated code would be:

int getHeight() const;

Since this is the default behavior for attributes in C/C++, this is also the default value for the property.

None - This value means that you are allowing the attribute value to be changed in the getter before it is returned.

SignatureAndReturnValue - This value means that not only is the getter function constant, but also the object returned to the calling function is constant. The generated code would be:

const class_2* getInfo() const;

This value modifies the generated code if the attribute is a class, but for primitive types the generated code would be the same as for the value "Signature".

Default = Signature

IsGuarded

The IsGuarded property specifies whether accessor and mutator operations are guarded. Guarded operations block if the object is not in a state in which it can be executed.

The possible values are as follows:

- none - Neither accessors nor mutators are guarded.
- mutator - Only mutators are guarded.
- all - Both accessors and mutators are guarded.

(Default = none)

IterType

When get/set functions are generated for an attribute with multiplicity > 1 (array), they take the array index as a parameter. By default, the software generates an index of type int.

The property IterType allows you to specify a different type for the array index.

To use a different type, just enter the name of the type as the value for this property.

This property can be set for individual attributes, or at higher levels, such as class, package, or project.

Default = Blank

Mutator

The Mutator property specifies the format of the names of mutator operations generated for attributes.

The default string, “set_$attribute:c”, means that if a mutator is generated for an attribute, the generated method name is set_attributeName().

(Default = "set_$attribute:c")

CGGeneral

The CGGeneral metaclass contains a property that controls canonical operations.

GeneratedCodeInBrowser

The GeneratedCodeInBrowser property specifies whether canonical operations (get/set) are added to the model and displayed in the browser. The possible values are as follows:

- Checked - Display automatically generated operations in the browser tree.
- Cleared - Do not display canonical operations.

(Default = Cleared)

IgnoreGuardedOperationVisibility

The IgnoreGuardedOperationVisibility property is a Boolean value that specifies whether the visibility of a guarded operation is ignored. This property is available at the project level.

This property was added for upgrade reasons only. It should not be used directly, and should not exist in models created with Rational Rhapsody version 4.0 and higher.

(Default = Checked)

InstallLayoutAs2.3

The InstallLayoutAs2.3 property specifies whether to use the generated file layout from Rational Rhapsody version 2.3.

Set this property at the project level.

(Default = Checked)

Class

The Class metaclass contains properties for implementing classes and objects.

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active classes. The possible values are as follows:

- A string - Specifies the message queue size for an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

(Default = empty string)

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects. The possible values are as follows:

- Any integer - Specifies that a stack of that size is allocated for active objects.
- An empty string (blank) - If not specified, the stack size is set in an operating system-specific manner, based on the value of the ActiveStackSize property for the framework.

(Default = empty string)

ActiveThreadName

The ActiveThreadName property indicates the real OS task or thread name. This property has an affect only when the class is set to active. This facilitates debugging in complex environments in which many threads are constantly being created and deleted on-the-fly. This property is effective for all targets. All strings entered must be enclosed in quotes (" "). The possible values are as follows:

- A string - Names the active thread.

- An empty string (blank) - The value is set in an operating system-specific manner, based on the value of the ActiveThreadName property for the framework.

(Default = empty string (OS selects thread name))

ActiveThreadPriority

The ActiveThreadPriority property specifies the priority of active class threads. The possible values are as follows:

- A string - Specifies thread priority of an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

(Default = empty string)

AdditionalCleanupCode

The AdditionalCleanupCode property is a MultiLine value that enables you to specify additional cleanup code. (Default = empty MultiLine)

AdditionalInitializationCode

The AdditionalInitializationCode property is a multiline value that enables you to specify additional initialization code. (Default = empty MultiLine)

AttributesAutoArrange

The AttributesAutoArrange property is a Boolean UI helper property. You should not modify this property directly. (Default = Checked)

CallUserInitRelations

The CallUserInitRelations property is a Boolean value that determines whether to call a user-defined initRelations() method. In Rational Rhapsody, initRelations() is called by the generated code (in the class constructors) even if you created your own initRelations() method.

Set this property to Cleared to disable the call to the user-defined initRelations().

(Default = Checked)

ComplexityForInlining

The ComplexityForInlining property specifies the upper bound for the number of lines in user code that are allowed to be inlined. User code is the action part of transitions in statecharts.

For example, using the value 3, all transitions with actions consisting of three lines or fewer of code are

automatically inlined in the calling function.

Inlining is replacing a function call in the generated code with the actual code statements that make up the body of the function. This optimizes the code execution at the expense of an increase in code size.

For example, increasing the number of lines that can be inlined from 3 to 5 has shortened the code execution time in some cases up to 10%.

This property applies only to the Flat implementation scheme for statecharts.

(Default = 3)

Concurrency

The Concurrency property specifies the concurrency of a class.

The CG::Operation::Concurrency property lets you specify that an operation should be protected by the object mutex (that is, only one access to the operation is allowed at any specific time). When a class has one or more operations with concurrency guarded, the class is becomes a guarded class.

The possible values for CG::Class::Concurrency are as follows:

- sequential - A sequential class maintains the single-threaded sequential protocol.
- active - An active class creates its own thread around which its operations are executed.

The possible values for CG::Operation::Concurrency are as follows:

- sequential - The operation is not guarded; access is sequential.
- guarded - The operation is protected by the object mutex.

(Default = sequential)

CreateImplicitDependencies

The CreateImplicitDependencies property is a Boolean value that prevents analysis of language types used by the class and its aggregates (as if all the types used by the class have their GenerateDeclarationDependency property set to Cleared).

(Default = Checked)

DeleteGlobalInstance

The DeleteGlobalInstance property specifies whether to delete a global instance of the class. Global instances are not deleted by default, because reaching a termination connector for a deleted global instance could cause a crash.

(Default = Cleared)

EmptyMemoryPoolCallback

The EmptyMemoryPoolCallback property specifies a name for the callback function that allocates more memory if the static pool is exhausted. This property provides support for static architectures.

(Default = empty string)

EmptyMemoryPoolMessage

The EmptyMemoryPoolMessage property specifies whether a message is output when the static memory pool is empty. This property provides support for static architectures. (Default = Checked)

Note: This property is only active during animation.

FileName

The FileName property specifies the name of the file to which code is generated for a class or package. This enables you to use a different name than the actual element name. For example, this feature can be of benefit if the class or package name is too long on 8.3 file systems.

The file name string can be either:

- A file name of your choice (including spaces)
- $name:n, where n is any digit between 1 and 9

The variable $name:n uses the name of the element as the basis for generating file names. The :n modifier specifies the length of the name. For example, if you specify $name:8, a class named YosemiteNationalPark would be generated into the files Yosemite.cpp and Yosemite.h.

If the FileName property is blank and the CG::Environment::IsFileNameShort property check box is checked, the code generator creates 8.3 file names by truncating the class name.

If the FileName property is defined as a file name longer than eight characters and the IsFileNameShort property check box is checked, the Checker reports an error. The long file name must be fixed prior to code generation. Each file name must be unique within the context of a single configuration.

If more than one class or package are generated to the same file name, they overwrite each other, causing compilation errors. To prevent this, the Checker determines whether multiple packages are directed to the same file before code is generated.

Setting the FileName property for an external base class (for example, to “BaseClass” (without the “.h”)) generates an #include of the base class in the specification file of the subclass.

You use the UseAsExternal property to mark a class as an external reference class. If the FileName property is not defined, you must add the appropriate code to the SpecificationProlog property for the subclass. (Default = empty string)

ForceReactive

The ForceReactive property is a Boolean value that specifies whether to generate a class as reactive, regardless of any other reactive criteria. A reactive class, that has a base class with this property check box checked, will not inherit directly from the framework reactive base class.

(Default = Cleared)

GenerateImplicitConstructors

The GenerateImplicitConstructors property is a Boolean value that specifies whether to generate implicit constructors. When the property check box is cleared, The software generates only user-specified constructors.

(Default = Checked)

GuardDestruction

If the property GuardDestruction is set to Checked, then the event queue cannot be destroyed in the middle of handling an event, and events cannot be consumed from the queue if the active/reactive class instance is in the process of being destroyed.

This property can be overridden or set at the site or project level to ensure that it is the default behavior.

(Default = Cleared)

ImplementStatechart

The ImplementStatechart property specifies whether to generate behavioral code for a reactive object. To use a statechart as documentation of behavior only (without generating behavioral code), do the following:

- Create a statechart for the object.
- Set the object's ImplementStatechart property to Cleared.
- Override the OMReactive::consumeEvent() method (in C++), or the RiCReactive.consumeEvent() function (in C), which implements the statechart.

This is one way of optimizing your statechart code. The default value for Rational Rhapsody Developer for Ada is Cleared; for all other languages, the default value is checked.

ImplicitDependencyToPackage

The ImplicitDependencyToPackage property is a Boolean value that determines whether the dependency from a class to its package is automatically generated.

(Default = Checked)

IncomeSignalMap

This property is used to map incoming SDL model signals to their SDL entry channels. The value of this property should not be modified by the user.

InitCleanUpRelations

The InitCleanUpRelations property specifies whether to generate initRelations() and cleanUpRelations() operations for sets of related global instances. This property applies only to composites and global relations.

(Default = Checked)

InitializerValue

The InitializerValue property enables you to initialize statically allocated objects and singletons. For example, for a singleton object A, if the InitializerValue property is set to {1,2,"abc"}, the resulting code is: struct A_t A = {1,2,"abc"}; You can access this property directly from the constructor window in the browser. (Default = empty string)

IsCompletedForAllStates

The IsCompletedForAllStates property specifies whether you can use the IS_COMPLETED(state) macro for all types of states. This macro is generally used in activity diagrams, but can also be used in statecharts.

The possible values are as follows:

- Checked - The IS_COMPLETED(state) macro can be used for all types of states.
- Cleared - The IS_COMPLETED(state) macro can be used only for states that have a Final state.

(Default = Cleared)

MaximumPendingEvents

Reserved for future use. (Default = –1)

OperationsAutoArrange

OperationsAutoArrange is a boolean property that determines the order in which operations are generated in the code. If set to checked, the order of the operations in the generated code is based on the Rational Rhapsody default order for the programming language being used.

If set to Cleared, the order of the operations in the generated code is the order specified by the user.

This means the order in which the user added the operations, unless the user overrode this order by making changes in the Edit Operations Order dialog or by using the Browser’s up/down buttons.

(Default = Checked)

OutcomeSignalMap

This property is used to map outgoing SDL model signals to their SDL exit channels. The value of this property should not be modified by the user.

ProtectStaticMemoryPool

The ProtectStaticMemoryPool property specifies whether to protect the static memory pool using an operating system mutex. This property helps support static architectures.

(Default = Checked)

ReactiveSimpleComposites

The ReactiveSimpleComposites property was added in Rational Rhapsody 5.2 for backwards compatibility. See the upgrade history on the support site for more information on this property.

RelationsAutoArrange

The RelationsAutoArrange property is a Boolean UI helper property. You should not modify this property directly. (Default = Checked)

StandardOperations

The StandardOperations property enables you to add template-based code (based on resolution of keywords) to a class or event.

Every standard operation is associated with a logical name. You define the logical name of a standard operation by overriding the StandardOperations property to add a comma-separated list of the names of the standard operations you want to define.

For every standard operation defined, you also need to specify an operation declaration and definition.

This is done by adding the following two properties to the site.prp file to specify the necessary function templates:

- LogicalName>Declaration - Specifies a template for the operation declaration
- LogicalName>Definition - Specifies a template for the operation implementation

For example, for a logical name of myOp, you would define the following property (using the site.prp file or the COM API(VBA)) :

Subject CG Metaclass Class Property myOpDeclaration MultiLine "" Property myOpDefinition MultiLine "" end

You add all of the properties to be associated with a standard operation to the site.prp file under their respective CG subject and metaclasses. All of these properties should have a type of MultiLine.

UseAsExternal

The UseAsExternal property specifies whether an object is referenced as an external object (one that was not generated in Rational Rhapsody). This property enables you to reference an external object in the model without generating code for it.

To prevent compilation errors with instrumentation when inheriting from an external base class (in C++), set the UseAsExternal property for the external base class to checked.

This prevents serialization code from being added to the base class. Similarly, the serialization operations are not called in the subclass.

Setting the FileName property for the base class (for example, to “BaseClass” without the “.h”) generates an #include of the base class in the specification file of the subclass. If the FileName property is not defined, you must add the appropriate code to the SpecificationProlog property for the subclass.

You can also use the UseAsExternal property to create template (parameterized) classes as follows:

- Create the template class outside of Rational Rhapsody.
- In the model, define a template class (for example, DataStore) as a placeholder for the external code defining the class.
- Set the UseAsExternal property for DataStore in the Rational Rhapsody model to checked.

Setting UseAsExternal to checked for the class prevents code from being generated for the external class when you generate code. You can still use the class in an object model diagram and inherit from it. If you want to inherit from an external template class, you can specify the external base class in the SpecificationProlog property for the subclass.

(Default = Cleared)

Component

The Component metaclass contains properties for implementing components.

CalculatePackageEventBaseId

The CalculatePackageEventBaseId property specifies how Rational Rhapsody calculates each package base event ID. This property is used to supports large and partially loaded models because you can specify a hash algorithm for the ID generation (using the property PackageEventBaseIdAlgorithm).

The possible values are as follows:

- OnCodeGeneration - Calculate the package base event ID during code generation.

- OnCreatePackage - Calculate the package event base ID when the package is created.

(Default = OnCreatePackage)

ComponentsSearchPath

The ComponentsSearchPath property specifies the name of related components. Adding the name of a component here causes the code generator to search the component for the file names of any related model elements that are not found in the original component. They are also added to the makefile search path.

Related components must have the same configuration name.

(Default = empty string)

InitializationScheme

The InitializationScheme property specifies how relations are initialized. This has implications for how relations are initialized across packages.

The possible values are as follows:

- ByPackage - Each package is responsible for its own initialization. The component's only responsibility is to declare an attribute for each package in the class.
- ByComponent - The component is responsible for initializing all global relations declared in all packages. This initialization is done with explicit calls in the component-class constructor for each package using initRelations() code (specified by the AdditionalInitialization property) and startBehavior().

Default = ByComponent

PackageCtrlDPMC

The PackageCtrlDPMC property was added in Rational Rhapsody 5.2 for backwards compatibility. See the upgrade history on the support site for more information on this property.

PackageEventBaseIdAlgorithm

The PackageEventBaseIdAlgorithm property specifies the algorithm Rational Rhapsody uses to calculate the package base event ID.

Note that this property value is only in effect when CG::Component::CalculatePackageEventBaseId is set to OnCreatePackage. The possible values are as follows:

- Fixed - The ID is based on random numbers.
- Hash - The ID is based on the name of the package.

(Default = Hash)

RelatedComponentsIncludePathInMakefile

The RelatedComponentsIncludePathInMakefile property was added in Rational Rhapsody 5.2 for backwards compatibility. See the upgrade history on the support site for more information on this property. (Default = Cleared)

UseDefaultNameForUnmappedElements

This UseDefaultNameForUnmappedElements property supports the use of an improved file name decision algorithm for naming files in code generation.

This property influences how file names are generated for elements that are out of the normal code generation scope of a component. By default, #include statements are not generated for elements that are out of the scope of the active component and its related components.

This prevents including non-existing files, which would cause compilation errors. The UseDefaultNameForUnmappedElements property provides control over how elements that are out of the known scope are generated.

The possible values are as follows:

- Checked - Use the default name for the element.
- Cleared - The name is an empty string. The result is that no #include statement can be generated for that file.

(Default = Checked)

Configuration

The Configuration metaclass contains properties for implementing configurations.

AddExplicitInitialInstancesToScope

The AddExplicitInitialInstancesToScope property enables you to include explicit initial instances as part of the scope for code generation. Beginning with version 5.0, Rational Rhapsody does not include explicit initial instances as part of the scope.

In other words, in explicit mode, code is not generated for a class just because it is in the list of initial instances for the configuration.

For existing models, Rational Rhapsody sets the CG::Configuration::AddExplicitInitialInstancesToScope property to checked at the project level to maintain the old behavior. This change enables you to use the list of initial instances to create instances that their classes defined in related components (libraries).

(Default = Checked)

AllowCollisionWithComponentName

The AllowCollisionWithComponentName property enables a check that prevents code generation when a class, actor, event, or global variable within the component scope has the same name as the component. In Rational Rhapsody Developer for Java, the check prevents generation of a class with the name Main "component."

This check was added because the software generates a class for the component.

When the model has global instances, multiple definitions of the same class are generated, one for the user class and the other for the generated component class. This means that if your model has elements and component with the same name, you must modify the class name, or the component name, in order to generate code.

You can disable the check by setting the AllowCollisionWithComponentName property to checked. However, if you do this, Rational Rhapsody protects you from redefinition and name collision at the code level. (Default = Cleared)

CodeGeneratorTool

The CodeGeneratorTool property specifies which code generation tool to use for the given configuration.

The possible values are as follows:

- External - Use the registered, external code generator.
- Internal - Use the software internal code generator.

The default value for Rational Rhapsody Developer for Ada is External; for all other languages, the default value is Internal.

ExternalGenerationTimeout

The ExternalGenerationTimeout property specifies how long, in seconds, Rational Rhapsody waits for the each class in the configuration scope to complete so you can once again make changes to the model. This property applies to both the full-featured external generator and makefile generator.

For example, if you set this property to 2 and you have 10 classes, Rational Rhapsody sets a timeout of 20. If the external code generator does not complete generation in this timeframe, Rational Rhapsody displays a message in the output window saying that the generator is not responding, and you are allowed to make changes to the model.

If you set this property to 0, Rational Rhapsody does not time out the generation session and waits for the code generator to complete its task - even if it takes forever. Rational Rhapsody waits for a notification from the full-featured external code generator or for the process termination of a makefile generator.

(Default = 0)

GenerateDirectoryPerModelComponent

The GenerateDirectoryPerModelComponent property specifies whether to generate a separate directory for each package in the component. The possible values are:

- Checked - Rational Rhapsody creates a separate directory for each package in the component. (This is the default.)
- Cleared - A separate directory is not created for each package.

ExternalGeneratorFileMappingRules

TheExternalGeneratorFileMappingRules property specifies whether the external code generator uses the same file mapping and naming scheme (mapping rules) as Rational Rhapsody.

If the mapping rules are different, the external generator must implement handlers to the GetFileName, GetMainFileName, and GetMakefileName events that Rational Rhapsody runs to get a requested file name and path.

The possible values are as follows:

- AsRhapsody - The external generator uses the same mapping rules as Rational Rhapsody.
- DefinedByGenerator - The external generator has its own mapping rules.

The default value for Ada is DefinedByGenerator; for all other languages, the default value is AsRhapsody.

GenerateForwardDeclarations

The GenerateForwardDeclarations property is a Boolean value that specifies whether forward declarations are generated.

(Default = Checked)

GeneratorExtraPropertyFiles

The GeneratorExtraPropertyFiles property launches the default Text Editor.

GeneratorRulesSet

The GeneratorRulesSet property enables you to specify your own rules set.

(Default = empty MultiLine)

GeneratorScenarioName

The GeneratorScenarioName property specifies the scenario name for the rule, if you write your own set of code generation rules. (Default = empty string)

LineWrapLength

The LineWrapLength property specifies the length of the code line generated during code generation. For example, if this property has the value 250, the generated code lines are wrapped to 250 characters. A value of 0 means that lines of code are not wrapped.

Note that code for the following elements are not wrapped, regardless of this property setting:

- User code parts (statechart actions and operation bodies)
- Element annotations
- Makefiles

(Default = 0)

MainGenerationScheme

The MainGenerationScheme property controls how the main procedure is generated. This property is required for compliance with MISRA® (Motor Industry Software Reliability Association) rules. The possible values are as follows:

- Full - The main procedure is generated as usual.
- UserInitializationOnly - The main contents generation is switched off and is replaced with only the initialization code field. This means that users can rewrite the main exactly as they want and will have to add any code that would normally be generated automatically by Rhapsody.
- For example, you would have to add the code for DefaultComponent_Init():

int main(int argc, char* argv[]) { /*#[configuration DefaultComponent::DefaultConfig */ // This is the initialization code added by the user /*#]*/ }

(Default = Full)

NotifyNeedOfModelCodeSync

The NotifyNeedOfModelCodeSync property is an enumerated type that controls whether Rational Rhapsody displays the message “Do you want to regenerate?” when you try to build a configuration. The possible values are as follows:

- Never - Never display the message.
- OnDynamicModelCodeAssociativity - Display the message only when the code generation is sensitive to changes (dynamic model-code associativity is on).
- Always - Always display the message.

(Default = OnDynamicModelCodeAssociativity)

PostFrameworkThreadSegment

The PostFrameworkThreadSegment property is a free text property added to the main() after the call to the

framework start() (OXF::start() in Rational Rhapsody Developer for C++).

In order for this code to be executed, you must set the property CG::Configuration::StartFrameworkInMainThread to Cleared. The default value for C and Java is an empty MultiLine; the default value for Ada and C++ is an empty string.

PreFrameworkInitCode

The PreFrameworkInitCode property specifies text that is added to the generated main() before the call to the framework initialization (OXF::init() in Rational Rhapsody Developer for C++).

This property was added to support additional customization features. For example, you could use this property to change the names of the OXFInit method arguments, argc and argv. (Default = empty MultiLine)

RemoveWhiteSpacesInBuildFile

The RemoveWhiteSpacesInBuildFile property determines whether spaces are removed from MULTI and INTEGRITY makefiles generated for models created before Rhapsody 6.0.

In models created using Rhapsody 6.0, spaces are not removed and this property is not available: the property is created automatically when you load a pre-Rhapsody 6.0 model.

You can add this property to the site.prp file, or directly change the factory.prp file, so you can access it for new models. (Default = Checked)

StartFrameworkInMainThread

The StartFrameworkInMainThread property is a Boolean value that determines whether the framework default event dispatcher should run in the main thread. If this is Cleared, the event dispatcher runs in a new thread. (Default = Checked)

StrictExternalElementsGeneration

The StrictExternalElementsGeneration property is a Boolean value that specifies whether to take advantage of information in modeled external elements during code generation. This property is used for backward compatibility (refer to the upgrade history on the support site for more information).

By default, this property check box is checked for models created before Rhapsody 5.2. For more information on modeling external elements, refer to the Rational Rhapsody Help.

(Default = Checked)

SupportExternalElementsInScope

The SupportExternalElementsInScope property is a Boolean value that specifies whether to include

external elements in the component scope. This property is used for backward compatibility (refer to the upgrade history on the support site for more information).

By default, this property check box is cleared for models created before Rhapsody 5.2. For more information on modeling external elements, refer to the Rational Rhapsody Help.

(Default = Cleared)

TrailingElseClause

The TrailingElseClause property can be used to add an else clause following an if/else if construct. The default value for this property is an empty string. If you replace this with a different string, the text you entered is placed within the braces of the else clause when the code is generated.

UnicodeEnvironment

The UnicodeEnvironment property makes the code generator assume that it is working in a Unicode environment (such as some Japanese versions). This property was added so non-English comments are generated correctly.

Default = Cleared

UseDescriptionTemplates

The UseDescriptionTemplates property is used for enabling template-based descriptions. This means the ability to define description template (with the <Lang>_CG:;Argument::DescriptionTemplate property) and instantiation of it using tags of specific elements. Code generation expands the templates with tag values and prints the full description in the generated code.

For Rational Rhapsody Developer for Java, the JavaDocProfile is loaded automatically for newly created Java projects with the default behavior to generate JavaDoc comments. To change the default for new Java projects, clear the check box for the UseDescriptionTemplates property. To disable the feature for a specific project, clear the Generate JavaDoc Comments check box on the Settings tab of the Configuration window. To enable JavaDoc comments on existing projects, load the JavaDocProfile.

Default = Checked

Dependency

The Dependency metaclass contains properties for implementing dependencies.

ConfigurationDependencies

The ConfigurationDependencies property enables you to control code generation usage of component

dependencies. Consider two components, A and B, and component A has a dependency with stereotype Usage to component B.

When you generate component A, this dependency is used by code generation for three reasons:

- To resolve file names (for include statements) of classes that are in the scope of B
- To add the B path to the makefile search path of A
- If B is a library and A is an executable, to add the B library to the link of A in the makefile

These tasks are based on the assumption that component B has a configuration with the same name as the active configuration of component A. The ConfigurationDependencies property enables you to mix and match the configuration by specifying configuration pairs, in the following format:

A config 1 nameB config 1 name, A config 2 nameB config 2 name,...

If you want to link A:debug_anim with B:debug_noAnim, set the ConfigurationDependencies property to "debug_anim:debug_noAnim." If the property holds the name of a configuration that does not exist, the configuration is ignored.

This functionality is especially useful with partial animation. For example, each component (A and B) has two configurations, debug_anim and debug_noAnim.

When the makefile is generated, Rational Rhapsody issues a warning for the unresolved dependency. This functionality is especially useful with partial animation. For example, each component (A and B) has two configurations, debug_anim and debug_noAnim.

(Default = empty string)

ForwardDeclarationPlacement

The ForwardDeclarationPlacement property allows positioning of the declaration either "before elements" or "at the top of the file."

(Default=BeforeElements)

GenerateRelationWithActors

The GenerateRelationWithActors property is an enumerated type that controls the generation of the dependency or relation to an actor (if it is a dependency or relation to an actor).

Control over generation of actors is done by the scope and by the appropriate check box in the Configuration Initialization tab.

The possible values for this property are as follows:

- Never - Never generate the dependency or relation.
- WhenActorIsGenerated - Generate the dependency or relation when the actor is generated.
- Always - Always generate the dependency or relation.

(Default = WhenActorIsGenerated)

PropagateImplementationToDerivedClasses

When the property CG::Dependency::UsageType is set to "Implementation", the .cpp file of the dependent element will contain an #include to the .h file of the element on which it depends. If the dependent element is a base class, then the .cpp files of its derived classes will also contain this #include. If you do not want this #include to be propagated to derived classes, set the value of the property PropagateImplementationToDerivedClasses to False.

(Default = Checked)

UsageType

The UsageType property specifies how a provider is to be made available to a dependent class or package if the Usage stereotype is attached to the dependency. The possible values are as follows:

- Existence - If the provider is a class, a forward class declaration is generated in the dependent.
- Implementation - An #include statement is generated in the implementation file of the dependent.
- Specification - An #include statement is generated in the specification file of the dependent.

(Default = Specification)

Event

The Event metaclass contains properties for implementing events.

AdditionalNumberOfInstances

The AdditionalNumberOfInstances property is a string that specifies the size of the local heap allocated for events when the current pool is full. Triggered operations use the event properties.

This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during runtime. All events are dynamically allocated during initialization.

Once allocated, a thread’s event queue remains static in size.

The possible values are as follows:

- An empty string (blank) - No additional memory is to be allocated when the initial memory pool is exhausted.
- n (a positive integer) - Specifies the size of the array allocated for additional instances. (Default = empty string)

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation.

The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property check box is cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property check box is cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property check box is cleared, all the arguments are not animated.
- If the AnimateArguments property check box is cleared, all the arguments are not animated, regardless of the specific argument Animate property settings. (Default = Checked)

AnimateArguments

The AnimateArguments property specifies whether to animate arguments to events or operations in animated sequence diagrams.

During instrumentation, Rational Rhapsody displays the values of actual parameters passed as arguments to events or operations in animated sequence diagrams using the C++ stream output operator.

For this to work, the argument must be of a built-in type that can be output to a stream, such as an int or a char*. If, however, the argument is of a complex, user-defined type that the compiler cannot serialize, then trying to animate it will cause compilation errors. In VxWorks, trying to animate arguments of type long unsigned int will also cause errors.

If you want to animate arguments of user-defined types defined either inside or outside of Rational Rhapsody, you must add to the framework an overloaded C++ stream output operator for the type.

(Default = Checked)

BaseNumberOfInstances

The BaseNumberOfInstances property is a string that specifies the size of the local heap memory pool allocated for either:

- Instances of the class (CPP_CG::Class)
- Instances of the event (CPP_CG::Event)

This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during run time. All instances of events are dynamically allocated during initialization.

Once allocated, a thread’s event queue remains static in size.

Triggered operations use the properties defined for events.

When the memory pool is exhausted, an additional amount, specified by the AdditionalNumberOfInstances property, is allocated.

Memory pools for classes can be used only with the Flat statechart implementation scheme.

The possible values are as follows:

- An empty string (blank) - Memory is always dynamically allocated.
- n (positive integer) - An array is allocated in this size for instances.

DeleteAfterConsumption

The DeleteAfterConsumption property enables you to create an event that cannot be deleted after it has been consumed. Such events are usually generated by interrupt handlers (see also GEN_ISR(event)).

The following is an example of an interrupt handler function that uses this type of event: void handler() { // Called upon an interrupt static Emergency e[10]; // Assumption: 10 events are enough static int i = 0; e[i].severity = 1; // Setting the event parameter itsSafetyController-gen(e[i]); // Do not use GEN here !! i=(i+1)%10; // Incrementing the buf pointer }

The possible values are as follows:

- Default - The default policy is to delete the event after consumption. However, using this value allows for possible future changes to the default policy. For example, events could be deleted some times, and not other times.
- Checked - The event is always deleted after consumption.
- Cleared - The event is never deleted after consumption.

(Default = Default)

EmptyMemoryPoolCallback

The EmptyMemoryPoolCallback property specifies a name for the callback function that allocates more memory if the static pool is exhausted.

This property provides support for static architectures. (Default = empty string)

EmptyMemoryPoolMessage

The EmptyMemoryPoolMessage property specifies whether a message is output when the static memory pool is empty. This property provides support for static architectures.

(Default = Checked)

Note: This property is only active during animation.

Generate

The Generate property specifies whether to generate code for a particular type of element. (Default = Checked)

GenEventIdAssignment

This property generates code for a specific event assignment. (Default = Checked)

Id

The Id property is a string that enables you to assign your own identification number to an event or triggered operation. The Id property assigns a permanent ID number to the event or triggered operation.

This supports the development of distributed systems in which an event or triggered operation must have the same ID in different parts of the system.

If a permanent ID is not assigned via this property, the number might change (for example, if the system has multiple components).

A check is performed before code is generated to verify that there are no conflicts between user-defined IDs and generated IDs. (Default = empty string)

IdNameScheme

The property IdNameScheme determines how the #define for the event is generated. The possible values are Full and Short.

If Full is selected, the #define for the event will include the entire hierarchy of the packages, for example: #define P1_P2_P3_Ev 2.

If Short is selected, the #define for the event will only include the name of the event and the name of the package it is under, for example: #define P3_Ev 2 The default value is Full. (Rhapsody versions prior to 6.1 MR-1 used this format.)

ProtectStaticMemoryPool

The ProtectStaticMemoryPool property specifies whether to protect the static memory pool using an operating system mutex. This property helps support static architectures. Triggered operations use the event's properties. (Default = Checked)

StandardOperations

The StandardOperations property enables you to add template-based code (based on resolution of keywords) to a class or event. Every standard operation is associated with a logical name. You define the logical name of a standard operation by overriding the StandardOperations property to add a comma-separated list of the names of the standard operations you want to define.

For every standard operation defined, you also need to specify an operation declaration and definition. This is done by adding the following two properties to the site.prp file to specify the necessary function templates:

- Declaration - Specifies a template for the operation declaration
- Definition - Specifies a template for the operation implementation

For example, for a logical name of myOp, you would define the following property (using the site.prp file or the COM API(VBA)) : Subject CG Metaclass Class Property myOpDeclaration MultiLine "" Property myOpDefinition MultiLine "" end

You add all of the properties to be associated with a standard operation to the site.prp file under their respective CG subject and metaclasses. All of these properties should have a type of MultiLine.

File

The File metaclass contains properties for implementing files.

AddToMakefile

The AddToMakefile property specifies whether a file is added to the makefile (and, therefore, built). This property supports modeling of flat source files (text without model elements).

It works in conjunction with the Generate property (also under CG::File).

This property supersedes the GenerateInMakefileOnly property. The following table shows the different settings for the two properties and the results.

Generate Value AddToMakefile Value File is Generated? File is Built? Checked Checked Yes Yes Checked Cleared Yes No Cleared Checked No Yes Cleared Cleared No No

(Default = Checked)

Footer

The Footer property specifies a multiline footer that is added to the end of generated files. The default footer template is as follows:

/********** File Path:$FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.

- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property lang_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

Generate

The Generate property specifies whether to generate code for a particular type of element. (Default = Checked)

Header

The Header property specifies a multiline header that is added to the top of all generated files. The default header template is as follows: /********** Rhapsody : $RhapsodyVersion Login : $Login Component : $ComponentName Configuration : $ConfigurationName Model Element : $FullModelElementName //! Generated Date : $CodeGeneratedDate File Path : $FullCodeGeneratedFileName **********/ For example: /********** Rhapsody : 4.0 Login : pie_man Component : BakedGood Configuration : DefaultConfig Model Element : Fairgoer::Simon //! Generated Date : Mon, 27, Nov 01 File Path : d:\projects\DefaultConfig\simple.h **********/ Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.

- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property lang_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

HeaderDirectivePattern

The HeaderDirectivePattern property controls the format of the #ifndef pattern in the generated .h files that prevents multiple definitions of the class.

The $FULLFILENAME_H keyword expansion is related to the actual directory structure and reflects the directory structure of the generated file (starting from the configuration directory).

When you use the keyword $FILENAME_H, the directory generated for the package is not reflected in the #ifndef and #define statements; when you use the $FULLFILENAME_H keyword, the directory is reflected in the statements.

Consider the following example: you have a package, P, that has a class, A. The C_CG/CPP_CG/JAVA_CG::Package::GenerateDirectory is set to Checked.

If you set the class property to $FILENAME_H, the following code is generated in A’s specification file:

#ifndef A_H #define A_H class A { ... }; #endif

If the property is set to $FULLFILENAME_H, the following code is generated in A’s specification file:

#ifndef P_A_H #define P_A_H class A { ... };

This property value is also used during roundtrip to ignore #define statements that match the pattern.

(Default = $FULLFILENAME_H)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated and external physical files for files in packages, components, and class/object/block/actor elements.

Note the following:

- This property is visible for files in packages, components, and class/object/block/actor elements.
- There is an ImpExtension property under lang_CG::Environment. However, the properties in the File metaclass have higher priority than those in the Environment metaclass.

(Default = empty string)

IncludeScheme

The IncludeScheme property specifies whether the path information is included in the file name in an #include statement. The possible values are as follows:

- RelativeToConfiguration - Include the relative path from the configuration directory in the include files.
- LocalOnly - The name of the include file is generated without the path information.

The property is first evaluated by the target file in the component model, if the file exists. If the file does not exist, the value is taken from the active configuration. This behavior enables you to specify the generation defaults for elements without files.

Note that this property affects only the files defined in the component , not the class. Therefore, you should define this property on the project, component, configuration, or file.

The property will not work correctly if you define for the class (even if the class is an element of the file, defined in the component).

(Default = RelativeToConfiguration)

InvokePostProcessor

The InvokePostProcessor property enables you to run a post-processing utility on the code generated by Rhapsody. For example, you could run a “beautify” program to get a specific coding style.

When this property value is not empty, Rational Rhapsody runs a process using the specified command.

You can specify the post-processing command on a single file or higher (folder, configuration, component, project, or site). You can specify the following keywords as part of the command:

- $file - The name of the generated file
- $projectPath - The current project root directory

Rhapsody generates code using the following sequence of events:

- Rhapsody generates code into a temporary file.
- If the target file already exists (because of a previous build), Rational Rhapsody compares the temporary file to the target file.
- If there are differences, the target file is replaced with the temporary file.
- If you specified a post-processor command, Rational Rhapsody runs the post processor on the temporary files. Any messages from the post-processor are displayed in the Output window.
- The temporary files are copied to the final location.

SpecExtension

The SpecExtension property specifies the extension that Rational Rhapsody appends to generated and external physical files for files in packages, components, and class/object/block/actor elements.

When you generate code for a dependency to a file with a non-standard extension, the generated #include includes the correct file extension, taken from this property.

Note that there is a SpecExtension property under lang_CG::Environment. However, the properties in the File metaclass have higher priority than those in the Environment metaclass.

See also the property ImpExtension. (Default = empty string)

General

The General metaclass contains a property used to support incremental code generation. See the Rational Rhapsody Help for more information on incremental code generation.

AbortOnModelChecker

The AbortOnModelChecker property specifies when to abort code generation, based on checks. The possible values are as follows:

- Errors - Abort code generation when errors are found prior to code generation. This is the default value.
- Warnings - Abort code generation when warnings or errors are found prior to code generation.

Most warnings are not started prior to code generation; the few that are started are related to problems that can result in incorrect behavior at run time.

(Default = Errors)

BuildErrorHandling

The BuildErrorHandling property allows forcing Rhapsody to treat all compiler errors as Code errors (versus element errors).

You can select the following choices from the drop-down list:

- "Model" means highlighting a model element on double-click compilation error on build.
- "Code" means highlighting a line in the code editor on double-click compilation error on build.

Note that "Model" sometimes highlights code lines if the model element cannot be found.

Default = Model

EnableProgressDialog

EnableProgressDialog is a Boolean property that allows you to specify whether or not Rational Rhapsody should display a progress dialog while code is being generated.

Default = Cleared

IncrementalCodeGenAcrossSession

The IncrementalCodeGenAcrossSession property is a Boolean value that controls the scope of the incremental code generation for the Rational Rhapsody session (either between sessions or only within the session).

If you set this to Cleared at the project level, code generation time stamps are not stored in the repository, and incremental code generation works only within a session.

(Default = Checked)

ReportToOutputWindow

This property disables some messages printed to the Output window during code generation. This property can improve performance during code generation particularly on Linux systems if the Output window is slow.

The values are Basic and Detailed. Detailed prints all messages, and Basic does not print the "Generating..." messages.

(Default = Detailed)

ShowLogViewAfterBuild

The ShowLogViewAfterBuild property allows you to specify that the Log tab should be brought to the front of the Output window after the completion of a build action, rather than the Build tab.

If this property is set to Checked, the Log tab is shown for all environments regardless of the value set for the property [lang]_CG::[environment]::UseNewBuildOutputWindow for the individual environments.

Default = Cleared

Generalization

The Generalization metaclass contains a property used to support generalization. See the Rational Rhapsody Help for more information on generalization.

Generate

The Generate property specifies whether to generate code for a particular type of element.

(Default = Checked)

ReportToOutputWindow

This property disables some messages printed to the Output window during code generation. This property can improve performance during code generation particularly on Linux systems if the Output window is slow.

The values are Basic and Detailed. Detailed prints all messages, and Basic does not print the "Generating..." messages.

(Default = Detailed)

Operation

The Operation metaclass contains properties for implementing operations.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property <lang>__CG::Type::AnimSerializeOperation.

The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property check box is cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property check box is cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property check box is cleared, all the arguments are not animated.
- If the AnimateArguments property check box is cleared, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = Checked

AnimateArguments

The AnimateArguments property specifies whether to animate arguments to events or operations in animated sequence diagrams.

During instrumentation, Rational Rhapsody displays the values of actual parameters passed as arguments to events or operations in animated sequence diagrams using the C++ stream output operator.

For this to work, the argument must be of a built-in type that can be output to a stream, such as an int or a char*.

If, however, the argument is of a complex, user-defined type that the compiler cannot serialize, then trying to animate it will cause compilation errors.

In VxWorks, trying to animate arguments of type long unsigned int will also cause errors.

If you want to animate arguments of user-defined types defined either inside or outside of Rational Rhapsody, you must do one of the following:

- Add to the framework an overloaded C++ stream output operator for the type, such as: ostream operator(sometype);
- Instantiate the template string2X(T t) with the type.
- Disable the AnimateArguments property by setting it to Cleared.

(Default = Checked)

Concurrency

The Concurrency property specifies the concurrency of a class. The CG::Operation::Concurrency property lets you specify that an operation should be protected by the object mutex (that is, only one access to the operation is allowed at any specific time).

When a class has one or more operations with concurrency guarded, the class becomes a guarded class.

The possible values for CG::Class::Concurrency are as follows:

- sequential - A sequential class maintains the single-threaded sequential protocol.
- active - An active class creates its own thread around which its operations are executed.

The possible values for CG::Operation::Concurrency are as follows:

- sequential - The operation is not guarded; access is sequential.
- guarded - The operation is protected by the object mutex.

(Default = sequential)

EnableInMethodBroker

The EnableInMethodBroker property specifies whether the operation should be considered in or excluded from the TestConductor MethodBroker.

Rhapsody does not allow you to call operations from outside of your application during model execution because it assumes that an application is triggered by external events or signals.

If you want to test subsystems or classes that do not react to external events, you must build a test environment in your model. A MethodBroker receives events and calls operations that need to be driven.

Rhapsody animation shows these operation calls as coming from the system border.

See the TestConductor documentation for more information about the MethodBroker.

You can exclude all operations of a class or package from the MethodBroker by setting the property EnableInMethodBroker at the class or package level.

(Default = Checked)

Generate

The Generate property specifies whether to generate code for a particular type of element. (Default = Full)

TriggeredOperationDefaultReturnValue

The property TriggeredOperationDefaultReturnValue allows you to define a default return value for a triggered operation. This default value is returned by the operation if, for some reason, the user code that sets the return value is not run.

Default = Blank

UseDefaultAttributeValues

The UseDefaultAttributeValues property is a Boolean value that specifies whether a constructor should initialize attributes according to the attribute's default value.

Set this property to Cleared to make a constructor ignore the attribute's default value.

(Default = Checked)

VariableLengthArgumentList

The VariableLengthArgumentList property specifies whether a variable length argument list is to be added to an operation's argument list.

If this is checked, a variable length argument list is added as the last argument for the operation.

For example, if the operation void f(int i) has this property set to checked, its generated declaration is void f(int i, ...).

(Default = Cleared)

Package

The Package metaclass contains properties for implementing packages.

AdditionalInitialization

The AdditionalInitializationCode property is a multiline value that enables you to specify additional initialization code.

(Default = empty MultiLine)

CallUserInitRelations

The CallUserInitRelations property is a Boolean value that determines whether to call a user-defined initRelations() method.

In Rhapsody, initRelations() is called by the generated code (in the package constructors) even if you created your own initRelations() method. Set this property to Cleared to disable the call to the user-defined initRelations().

(Default = Checked)

FileName

The FileName property specifies the name of the file to which code is generated for a class or package. This enables you to use a different name than the actual element name.

For example, this feature can be of benefit if the class or package name is too long on 8.3 file systems. The file name string can be either:

- A file name of your choice (including spaces)
- $name:n, where n is any digit between 1 and 9

The variable $name:n uses the name of the element as the basis for generating file names. The :n modifier specifies the length of the name.

For example, if you specify $name:8, a class named YosemiteNationalPark would be generated into the files Yosemite.cpp and Yosemite.h.

If the FileName property is blank and the CG::Environment::IsFileNameShort property check box is checked, the code generator creates 8.3 file names by truncating the class name.

If the FileName property is defined as a file name longer than eight characters and the IsFileNameShort property check box is checked, the Checker reports an error.

The long file name must be fixed prior to code generation. Each file name must be unique within the context of a single configuration.

If more than one class or package are generated to the same file name, they overwrite each other, causing compilation errors. To prevent this, the Checker determines whether multiple packages are directed to the same file before code is generated.

Setting the FileName property for an external base class (for example, to “BaseClass” without the “.h”)

generates an #include of the base class in the specification file of the subclass.

You use the UseAsExternal property to mark a class as an external reference class.

If the FileName property is not defined, you must add the appropriate code to the SpecificationProlog property for the subclass.

(Default = empty string)

GeneratePackageCleanup

The GeneratePackageCleanup property determines when the system should perform a clean-up operation after generating a Package. The possible values are Always, Never, and Smart.

GeneratePackageCode

The GeneratePackageCode property specifies whether to generate package code. This property supports “smart generation” of the package files. This property is ignored for COM/CORBA and Animation. The possible values are as follows:

- Always - Always generate package files.
- Never - Never generate package files.
- Smart - Generate package files only when the package contains elements that will produce meaningful code.

(Default = Smart)

GeneratePackageInitialization

The GeneratePackageInitialization property determines when the system should initialize package generation. The possible values are Always, Never, and Smart.

GenerateWithAggregates

The GenerateWithAggregates property determines whether all classes owned by a package are added along with the package when you add a new package to a particular scope.

(Default = Checked)

ImplicitDependencyToPackage

The property ImplicitDependencyToPackage allows you to suppress the generation of an #include to a parent package of a package. If set to False, Rational Rhapsody will not generate an #include to the parent package.

Default = Checked

InitCleanUpRelations

The InitCleanUpRelations property specifies whether to generate initRelations() and cleanUpRelations() operations for sets of related global instances. This property applies only to composites and global relations.

(Default = Checked)

InstancesAutoArrange

The InstancesAutoArrange property is a Boolean UI helper property. You should not modify this property directly. (Default = Checked)

SelfInit

The SelfInit property specifies whether to automatically initialize global instance variables. (Default = Cleared)

SynthesizeClassDependencies

The SynthesizeClassDependencies property controls when Rhapsody generates forward declarations and include statements. The possible values are as follows:

- All - By default, Rational Rhapsody generates forward declarations to the classes owned by a package in the package specification file, and generates the includes in the package implementation file.
- ByUsage - Rational Rhapsody generates forward declarations and include files only for the classes to which the package has an explicit dependency (with the Usage stereotype).

(Default = All)

UseAsExternal

The UseAsExternal property specifies whether an object is referenced as an external object (one that was not generated in Rational Rhapsody). This property enables you to reference an external object in the model without generating code for it.

To prevent compilation errors with instrumentation when inheriting from an external base class (in C++), set the UseAsExternal property for the external base class to checked. This prevents serialization code from being added to the base class.

Similarly, the serialization operations are not called in the subclass. Setting the FileName property for the base class (for example, to "BaseClass," without the ".h") generates an #include of the base class in the specification file of the subclass.

If the FileName property is not defined, you must add the appropriate code to the SpecificationProlog property for the subclass. You can also use the UseAsExternal property to create template (parameterized) classes as follows:

- Create the template class outside of Rational Rhapsody.
- In the Rational Rhapsody model, define a template class (for example, DataStore) as a placeholder for the external code defining the class.
- Set the UseAsExternal property for DataStore in the Rational Rhapsody model to checked.

Setting UseAsExternal to checked for the class prevents code from being generated for the external class when you generate code. You can still use the class in an object model diagram and inherit from it.

If you want to inherit from an external template class, you can specify the external base class in the SpecificationProlog property for the subclass. Note that when you set this property at the package level, all the aggregates automatically become external as well. (Default = Cleared)

Relation

The Relation metaclass contains properties for implementing relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname-insert(map$keyType,$target::value_type($keyName,$item))(Default = add$cname:c)*

AddComponentHelpersGenerate

The AddComponentHelpersGenerate property is a Boolean value that specifies whether to generate the helper method for a symmetric composite-part relationship, where the part multiplicity is greater than 1.

This property enables you to control the helper method (such as _addItsX()), which is used to connect a part X to its composite.

(Default = Checked)

AddGenerate

The AddGenerate property specifies whether to generate an Add() operation for relations. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

AddHelpersGenerate

The AddHelpersGenerate property is an enumerated type that enables you to control the relation helper methods, such as _addItsX() and __addItsX().

The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the CG::Relation::AddGenerate property.

(Default = True)

Clear

The Clear property specifies the name of an operation that removes all items from a relation.

For example, using the boilerplate clear$cname:c for a relation called itsServer, Rational Rhapsody would generate a public operation with the following signature: void clearItsServer(); (Default = clear$cname:c)

ClearGenerate

The ClearGenerate property specifies whether to generate a Clear() operation for relations. Setting this property to Cleared is one way to optimize your code for size. (Default = Checked)

ClearHelpersGenerate

The ClearHelpersGenerate property is an enumerated type that enables you to control the relation helper methods, such as _clearItsX() and __clearItsX(). The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the CG::Relation::ClearGenerate property.

(Default = True)

Containment

The Containment property specifies how to represent relational objects as members of classes. The possible values are as follows:

- Value - The container actually lives inside the class. For example: OMCollectionServer* itsServer;
- Reference - The class maintains a pointer to the collection. For example: OMCollectionServer** itsServer;

See also the properties CreateStatic and InitStatic under ContainerType>::RelationType>. (Default = Value)

CreateComponent

The CreateComponent property specifies the name of an operation that creates a new component in a composite class.

For example, using the boilerplate new$cname:c for a composite class with a component called Server, Rational Rhapsody generates a public operation with the following signature:

Server* newSv();

(Default = new$cname:c)

CreateComponentGenerate

The CreateComponentGenerate property specifies whether to generate a CreateComponent operation for composite objects. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

CreateComponentUsingIndex

This property is included in the backward compatibility profile for version 6.1 MR-1 of Rational Rhapsody (CGCompatibilityPre61M1[lang]).

In version 6.1 MR-1 of Rational Rhapsody, improvements were made to the initialization code generated for parts with bounded multiplicity implemented as StaticArray (for example, C* itsC[5]).

These improvements avoid the redundant search for free locations in the array inside the composite create operation (e.g., newItsC()). This is done by passing the index to the create operation from the external loop in initRelations().

Since this represented a change to the create operation signature and behavior (e.g., newItsC() replaced by newItsC(int i)), the change was disabled for pre-6.1 MR-1 models. This was accomplished by setting the value of the property CG::Relation::CreateComponentUsingIndex to False.

The property applies to RiC, RiC++, and RiJ.

DeleteComponent

The DeleteComponent property specifies the name of an operation that deletes a component from a composite class.

For example, using the boilerplate delete$cname:c for a composite class with a component called Server, Rational Rhapsody generates a public operation with the following signature:

void deleteSv(Server* p_Server);

(Default = delete$cname:c)

DeleteComponentGenerate

The DeleteComponentGenerate property specifies whether to generate a DeleteComponent() operation for composite objects. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

Dependency

The Dependency property specifies where to include the specification file (.h) of the dependent class in a relationship between two classes. The possible values are as follows:

- Weak - The specification file of the dependent class is included in the implementation file of the independent class with a forward declaration in the specification file.
- Strong - The specification file of the dependent class is included in the (.h) file of the independent class.

(Default = Weak)

FilledDiamondInitializationScheme

The FilledDiamondInitializationScheme specifies how a filled-diamond relation is initialized. The possible values are as follows:

- Automatic - from the default
- ByUser - manually by the user

(Default = ByUser)

FilledDiamondScheme

The FilledDiamondScheme property specifies how a filled-diamond relation is implemented. The possible values are as follows:

- Composition - Implement the relation as a composition. The automatically generated code will create and destroy the relation instances.
- Aggregation - Implement the relation as an aggregation. You are responsible for creating and deleting the relation elements.

For models that were created in versions prior to Rational Rhapsody 4.0, this property is automatically set to Aggregation when you load the model into Rational Rhapsody 4.0. This is due to the fact that only aggregation was supported in the previous versions of Rational Rhapsody.

(Default = Aggregation)

Find

The Find property specifies the name of an operation that locates an item among relational objects.

For example, using the boilerplate find$cname:c for a relation called itsServer, Rational Rhapsody generates a public operation with the following signature:

int findItsServer(Server* p_Server)const;

(Default = find$cname:c)

FindGenerate

The FindGenerate property specifies whether to generate a Find() operation for relations. Setting this property to Cleared is one way to optimize your code for size. (Default = Cleared)

Generate

The Generate property specifies whether to generate code for a particular type of element. To optimize space, do not implement links to global or singleton objects by setting this property to Cleared.

(Default = Checked)

GenerateRelationWithActors

The GenerateRelationWithActors property is an enumerated type that controls the generation of the dependency or relation to an actor (if it is a dependency or relation to an actor).

Control over generation of actors is done by the scope and by the appropriate check box in the Configuration Initialization tab.

The possible values for this property are as follows:

- Never - Never generate the dependency or relation.
- WhenActorIsGenerated - Generate the dependency or relation when the actor is generated.
- Always - Always generate the dependency or relation.

(Default = WhenActorIsGenerated)

Get

The Get property specifies the name of an operation that retrieves the relation currently pointed to by the iterator.

For example, using the boilerplate get$cname:c for a relation called itsServer, Rational Rhapsody generates an operation with the following signature: OMIteratorServer* getItsServer()const; The Get

property under a container metaclass (for example, under RiCContainers::Scalar) specifies the template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The keyword $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = get$cname:c)

GetEnd

The GetEnd property specifies the name of an operation that points the iterator to the last item in a collection. This property is used with the Standard Template Library.

For example, using the boilerplate get$cname:cEnd for a relation named itsServer, Rational Rhapsody generates a public operation with the following signature: vectorServer*::const_iterator Server::getItsServerEnd() const; This function points the iterator to the last item in the collection by testing for the following condition: iter == getItsServerEnd()

(Default = get$cname:cEnd)

GetEndGenerate

The GetEndGenerate property specifies whether to generate a GetEnd() operation for relations when using STL. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

GetGenerate

The GetGenerate property specifies whether to generate accessor operations for relations. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

IgnoreQualifierOnBlackDiamond

If this the IgnoreQualifierOnBlackDiamond property is selected, it instructs the system to ignore the qualifier on a black diamond.

(Default = Cleared)

Implementation

The Implementation property specifies how to implement concrete relations. The possible values are as follows:

- Default - Rational Rhapsody automatically selects the appropriate container.
- Scalar - Implement a to-one relation as a pointer to a single object.
- StaticArray - Implement a to-many relation as a static array of fixed size.
- Fixed - Implement a to-many relation whose multiplicity is a number greater than 1 as an array of fixed size.
- BoundedOrdered - Implement a to-many relation whose multiplicity is a number greater than 1 and whose Ordered property check box is checked as a list.
- BoundedUnordered - Implement a to-many relation whose multiplicity is a number >1 and whose Ordered property check box is cleared as a collection.
- UnboundedOrdered - Implement a to-many relation with an upper limit of * and whose Ordered property check box is checked as a list.
- UnboundedUnordered - Implement a to-many relation with an upper limit of * and whose Ordered property check box is cleared as either a list or a collection, depending on the language.
- EmbeddedScalar - Implement a to-one composite relation as an embedded object (C and C++ only).
- EmbeddedFixed - Implement a to-many composite relation as an array of embedded objects (C and C++ only).
- Qualified - Implement a to-many qualified relation as a map or hashtable, depending on the language.
- User - User-defined containers. In this case, you must provide a complete definition of all properties relevant to the following metaclasses:
- Java(1.1)Containers::User (Java)
- Java(1.2)Containers::User (Java)
- OMContainers::User (C++)
- OMUContainers::User (C++)
- RiCContainers::User (C)
- STLContainers::User (C++ with STL)

To optimize performance and space, implement relations as real arrays whenever possible by setting this property to StaticArray or Scalar instead of using template-based implementations (such as OMCollection, OMList, and OMMap).

The array implementation is simpler and faster.

To create an optimized implementation of the relation to be used by the code generator, add the interface to the MContainers::User metaclass, then set the Implementation property for the relation to User.

(Default = Default)

InstanceBasedLinking

The InstanceBasedLinking property enables you to connect a relation with variant multiplicity between instances.

Rhapsody can connect a relation between instances based on the object model diagram information. It connects instances based on the following principles:

- The instances to connect are owned by the same package or composite class.
- The multiplicity of the relation and instances match (that is, all the instances is connected based on the relation). As shown in the following figure, the match is calculated as n*M = N*M, where in a directional relation from A to B, N is set to 1.

In previous versions of Rational Rhapsody, the instances were connected only when M and N were constants. You can connect an instance even if the relation multiplicity is unconstrained (*) or variant (such as 0...5).

Connecting instances over a relation with variant multiplicity is based on the following principles:

- The InstanceBasedLinking property check box is checked.
- If M = *, M is considered to have the same value as m.
- If M = x..y:
- m must be greater than x.
- M is considered to have the same value as min(y,m).

IsConst

The IsConst property specifies whether accessor operations are const member functions. It can also be used to specify whether the return type of such an operation is a const. The possible values are as follows:

- None - not a const member function
- Signature - is a const member function
- SignatureAndReturnType - is a const member function, and the return type is a const

The default value is Signature.

IsGuarded

The IsGuarded property specifies whether accessor and mutator operations are guarded. Guarded operations block if the object is not in a state in which it can be executed.

The possible values are as follows:

- none - Neither accessors nor mutators are guarded.
- mutator - Only mutators are guarded.
- all - Both accessors and mutators are guarded.

(Default = none)

Ordered

The Ordered property specifies whether relations are ordered. (Default = Cleared)

Remove

The Remove property specifies the name of an operation that removes an item from a relation.

For example, using the boilerplate remove$cname:c for a relation called itsServer, Rational Rhapsody generates a public operation with the following signature: void removeItsServer(Server* p_Server);

(Default = remove$cname:c)

RemoveComponentHelpersGenerate

The RemoveComponentHelpersGenerate property is a Boolean value that enables you to control the relation helper methods, such as _removeItsX() and __removeItsX().

(Default = Checked)

RemoveGenerate

The RemoveGenerate property specifies whether to generate a Remove() operation for relations. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

RemoveHelpersGenerate

The RemoveHelpersGenerate property is an enumerated type that enables you to control the relation helper methods, such as removeItsX() and __removeItsX().

The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the CG::Relation::RemoveGenerate property.

(Default = True)

SafeInitScalar

The SafeInitScalar property specifies whether to initialize scalar relations as null pointers.

Consider the following C example: For a relation between an object A and a single object B, the pointer to itsB is initialized to NULL in the initializer for A, as follows:

A_Init() { me-itsB = NULL; initRelations(me); }

(Default = Checked)

Set

The Set property specifies the name of the mutator generated for scalar relations.

For example, using the template set$cname:c for a scalar relation called itsServer, Rational Rhapsody generates a public operation with the following signature:

void setItsServer(Server* p_Server);

(Default = set$cname:c)

SetComponentHelpersGenerate

The SetComponentHelpersGenerate property is a Boolean value that enables you to control the relation helper methods, such as _setItsX() and __setItsX().

(Default = Checked)

SetGenerate

The SetGenerate property specifies whether to generate mutators for relations. Setting this property to Cleared is one way to optimize your code for size.

(Default = Checked)

SetHelpersGenerate

The SetHelpersGenerate property is an enumerated type that enables you to control the relation helper methods, such as _setItsX() and __setItsX().

The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the CG::Relation::SetGenerate property.

(Default = True)

Statechart

The Statechart metaclass contains a property to control statecharts.

EventConsumptionScheme

The EventConsumptionScheme property controls whether multiple exits from state_dispatchEvent operations are allowed. An exit is needed only for AND state dispatchEvent functions.

This property is required for compliance with MISRA (Motor Industry Software Reliability Association) rules.

The possible values are as follows:

- Default - Multiple exits are allowed. This is normal pre-V4.2 behavior.
- SingleExitPoint - There is a single exit point.

For example, the following code shows the normal pre-V4.2 behavior (Default): if(operation_dispatchEvent(me, id) 0) { res = eventConsumed; if(!IS_IN(me, Dishwasher_active)) return res; } if(service_dispatchEvent(me, id) 0) { res = eventConsumed; if(!IS_IN(me, Dishwasher_active)) return res; } return res; The following example shows the code generated when this property is set to SingleExitPoint: RiCBoolean dispatchDone = FALSE; if (operation_dispatchEvent(me, id) 0) { res = eventConsumed; if(!IS_IN(me, Dishwasher_active)) { dispatchDone = TRUE; } } if (!dispatchDone) { if (service_dispatchEvent(me, id) 0) { res = eventConsumed; if(!IS_IN(me, Dishwasher_active)) { dispatchDone = TRUE; } } } return res;

FlatStateType

The FlatStateType property enables you to specify another data type to use when defining attributes for flat statecharts.

By default, Rational Rhapsody creates these attributes using int values. You can set this property at or above the statechart level. (Default = empty string)

LocalTerminationSemantics

The LocalTerminationSemantics property specifies whether activity diagram mode is enabled for local termination of behavior.

Local termination refers to the termination of an activity rather than an instance when a Termination connector is reached in an activity diagram or statechart.

There are two termination types:

- Local termination - When the Termination connector is inside a composite state, it is considered a Final state, which terminates the activity represented by the composite state, but not the instance performing the activity.
- There are two types of local termination:
- Statechart mode - Local termination semantics can be applied only to an Or state that has a Final state as a substate.
- Activity diagram mode - Local termination semantics can be applied to any Or state, even one without a Final state.

- Global termination - When the Termination connector is inside the top state, it is considered a Termination state, which terminates the state machine, causing the instance to be destroyed.

If this property is Cleared, statechart mode is enabled for local termination. If it is checked, activity diagram mode is enabled for local termination. (Default = Cleared)

StateInOperationReturnType

When code is generated for statecharts, a number of operations are generated for testing whether the system is in a given state. By default, these operations return an int in RiC and a bool in RiC++. The property StateInOperationReturnType allows you to specify a different return type for these operations. This can be used to ensure that the generated code meets standards such as MISRA.

Default = Blank

Type

The Type metaclass contains properties that control data types.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation.

The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property check box is cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property check box is cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property check box is cleared, all the arguments are not animated.
- If the AnimateArguments property check box is cleared, all the arguments are not animated, regardless of the specific argument Animate property settings.

For most of the Animate properties, the possible values are checked and Cleared. However, the property CG::Type::Animate has three possible values:

- True - The code generator analyzes the data type and instruments it according to its type. If the type is unknown, its address is converted to void*.
- False - Disable the generation of animation calls.
- Force - Generate animation calls. The code generator generates either standard (x2String or string2X) calls, or those calls defined in the AnimSerializeOperation and AnimUnserializeOperation properties.

If you define a type that cannot be instrumented, you should declare the instrumentation operation’s name and create the operations manually, or define properties to allow generation of these operations.

(Default = Force)

EnumerationAsTypedef

The EnumerationAsTypedef property is a Boolean value that determines whether enumeration composite types are printed with typedef statements. (Default = Checked)

Generate

The Generate property specifies whether to generate code for a particular type of element. (Default = Checked)

GenerateDeclarationDependency

When you provide code for the declaration of a "Language" type, or specify the type of an element such as an attribute by providing your own code in the C++ Declaration field (rather than selecting a type from the list of existing types), Rational Rhapsody cannot know with 100% certainty what type your declaration is dependent upon.

In such cases, Rational Rhapsody parses the code you entered, and searches the model for the type referenced in the declaration. It will take the first type it encounters with this name, and the code generated will include a dependency upon this type.

In cases where you have defined types with the same name in different packages, this type-searching behavior by Rhapsody may result in #include statements that do not reflect what you had intended. To avoid this, you can set the value of the property GenerateDeclarationDependency to False, and Rhapsody will not generate any dependency in the code to reflect your declaration. You can then explicitly create the correct dependency in the model so that the generated code includes all the necessary dependencies.

For example:

If you have defined an enum called "color" in the package Basic_types, and then have a package called Vehicles with a class named Car, and that class contains an attribute named exterior_color of type "color", Rational Rhapsody will by default generate the necessary #include in the code for class Car:

#include "Basic_types.h"

If, however, you set the value of the property GenerateDeclarationDependency to False, the generated code will not include the necessary dependency and you will have to provide it directly.

Keep in mind that the property appears under the metaclass Type, but in terms of code generation it affects the code generated for specific attributes that are of a given type. So you have to open the features dialog for the relevant attributes and change the value of the property there.

Default = Checked

Implementation

The Implementation property enables you to specify how Rhapsody generates code for a given element (for example, as a simple array, collection, or list).

When this property is set to Default and the multiplicity is bounded (not *) and the type of the attribute is not a class, code is generated without using the container properties (as in previous versions of Rational Rhapsody).

Note that Rational Rhapsody generates a single accessor and mutator for an attribute, as opposed to relations, which can have several accessors and mutators.

(Default = Default)

UseAsExternal

The UseAsExternal property specifies whether an object is referenced as an external object (one that was not generated in Rational Rhapsody).

This property enables you to reference an external object in the model without generating code for it.

To prevent compilation errors with instrumentation when inheriting from an external base class (in C++), set the UseAsExternal property for the external base class to checked.

This prevents serialization code from being added to the base class. Similarly, the serialization operations are not called in the subclass.

Setting the FileName property for the base class (for example, to "BaseClass," without the ".h") generates an #include of the base class in the specification file of the subclass.

If the FileName property is not defined, you must add the appropriate code to the SpecificationProlog property for the subclass.

You can also use the UseAsExternal property to create template (parameterized) classes as follows:

- Create the template class outside of Rational Rhapsody.
- In the Rational Rhapsody model, define a template class (for example, DataStore) as a placeholder for the external code defining the class.
- Set the UseAsExternal property for DataStore in the Rational Rhapsody model to checked.

Setting UseAsExternal to checked for the class prevents code from being generated for the external class when you generate code. You can still use the class in an object model diagram and inherit from it.

If you want to inherit from an external template class, you can specify the external base class in the SpecificationProlog property for the subclass.

(Default = Cleared)

Collaboration_Diagram

The Collaboration_Diagram subject contains the following metaclasses:

- AssociationRole
- AutoPopulate
- CollaborationDiagramGE
- Depends
- Messages

AssociationRole

The AssociationRole metaclass contains a property that controls the display of associations in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,0,0)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Bottom-Top

Classifier

The Classifier metaclass contains properties that control the display of classifiers in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,0,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,0,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

ClassifierActor

The ClassifierActor metaclass contains properties that control the display of classifier actors in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,0,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,0,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

CollaborationDiagramGE

The CollaborationDiagramGE metaclass contains a property that controls the display of collaboration diagrams in Rational Rhapsody.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

CollMessage

The CollMessage metaclass contains properties that control the display of link messages in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,0,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Comment

The Comment metaclass contains properties that control the appearance of comments in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

Constraint

The Constraint metaclass contains properties that control the appearance of comments in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Depends

The Depends metaclass contains properties that control the appearance of dependency relation lines in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,0,255)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,255)

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

Messages

Contains properties that affect the display of messages in collaboration diagrams.

ShowNumbering

ShowNumbering is a boolean property that determines whether or not Rational Rhapsody displays the sequence numbers for messages in a collaboration diagram.

Default = Checked

MultiObj

The MultiObj metaclass contains properties that control the display of multiple objects in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,0,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,0,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Note

The Note metaclass contains properties that control the appearance of notes in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,128,64)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,128,255)

Requirement

The Requirement metaclass contains properties that control the appearance of requirements in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

ReverseCollMessage

The ReverseCollMessage metaclass contains properties that control the display of reverse link messages in collaboration diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,0,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

(Default = None)

COM

The COM subject contains properties for allowing mixed, distributed applications and objects to find and interact with each other over a network. For more information on COM properties, see the MSDN Online Library (http://msdn.microsoft.com/library/). The COM subject is available in Rational Rhapsody Developer for C++ only. The subject COM contains properties under the following metaclasses:

- Argument
- Attribute
- Class
- coclass
- Configuration
- IDL
- Interface
- Library
- Operation
- Relation

Argument

The Argument metaclass contains properties that control COM arguments. Many of these properties are standard MSDN COM properties.

AppendToClause

The AppendToClause property is an optional verbatim property that enables you to add free text to be generated into the end of the description clause, before the closing bracket.

(Default = empty string)

defaultvalue

The defaultvalue property is a standard MSDN COM property. This property enables specification of a default value for a typed optional parameter.

(Default = empty string)

first_is

The first_is property is a standard MSDN COM property. This property specifies the index of the first array element to be transmitted. (Default = empty string)

ignore

The ignore property is a standard MSDN COM property. This property designates that a pointer contained in a structure or union (and the object indicated by the pointer) is not transmitted, and is restricted to pointer members of structures or unions.

(Default = Cleared)

iid_is

The iid_is property is a standard MSDN COM property. This property specifies the IID of the COM interface pointed to by an interface pointer.

(Default = empty string)

last_is

The last_is property is a standard MSDN COM property. This property specifies the index of the last array element to be transmitted. When the specified index is zero or negative, no array elements are transmitted.

(Default = empty string)

lcid

The lcid property is a standard MSDN COM property. This property indicates that the parameter is a locale ID (LCID).

(Default = Cleared)

length_is

The length_is property is a standard MSDN COM property. This property specifies the number of array elements to be transmitted. Specify a non-negative value.

(Default = empty string)

max_is

The max_is property is a standard MSDN COM property. This property designates the maximum value for a valid array index.

(Default = empty string)

optional

The optional property is a standard MSDN COM property. This property specifies an optional parameter.

(Default = Cleared)

pointer

The pointer property is a standard MSDN COM property. Explicit pointer attributes are applied to the pointer at the definition or use site.

(Default = empty string)

readonly

The readonly property is a standard MSDN COM property. This property specifies whether the parameter is read only.

(Default = Cleared)

retval

The retval property is a standard MSDN COM property. This property designates the parameter that receives the return value of the member. (Default = Cleared)

size_is

The size_is property is a standard MSDN COM property. This property specifies the size of memory allocated for sized pointers, sized pointers to sized pointers, and single- or multidimensional arrays.

(Default = empty string)

string

The string property is a standard MSDN COM property. This property specifies a string.

(Default = Cleared)

Attribute

The Attribute metaclass contains properties that control COM attributes.

AppendToClause

The AppendToClause property is an optional verbatim property that enables you to add free text to be generated into the end of the description clause, before the closing bracket.

(Default = empty string)

bindable

The bindable property is a standard MSDN COM property. This property indicates that the property supports data binding.

(Default = Cleared)

call_as

The call_as property is a standard MSDN COM property. This property enables mapping a function, which cannot be called remotely, to a remote function.

(Default = empty string)

defaultbind

The defaultbind property is a standard MSDN COM property. This property indicates the single, bindable property that best represents the object.

(Default = Cleared)

defaultcollelm

The defaultcollelm property is a standard MSDN COM property. This property allows for optimization of code.

(Default = Cleared)

helpcontext

The helpcontext property is a standard MSDN COM property. This property sets the context in the Help file.

(Default = empty string)

helpstring

The helpstring property is a standard MSDN COM property. This property specifies the text that describes the element to which it applies, such as a library, dispinterface, or coclass.

(Default = empty string)

hidden

The hidden property is a standard MSDN COM property. This property indicates that the item exists, but should not be displayed in a user-oriented browser.

(Default = Cleared)

id

The id property specifies the ID used for get/put methods written to the COM interface file. Relations between COM interfaces are treated like COM interface attributes, and the required get/put methods are written to the COM interface file.

(Default = empty string)

immediatebind

The immediatebind property is a standard MSDN COM property. This property allows individual, bindable, properties on a form to specify this behavior. When this bit is set, all changes is notified.

(Default = Cleared)

implementation

The implementation property is an enumerated type that specifies which accessor and mutator methods (get/put/putref) should be generated for a COM attribute in the COM interface. The possible values are as follows:

- propget - Generate an accessor only.
- propput - Generate a mutator only.
- propputref - Generate a by-reference mutator only.
- propgetpropput - Generate both an accessor and a mutator (the default value).
- propgetpropputRef - Generate both an accessor and a by-reference mutator.

(Default = propgetpropput)

local

The local property is a standard MSDN COM property. This property specifies to the MIDL compiler that an interface or a function is not remote.

(Default = Cleared)

nonbrowsable

The nonbrowsable property is a standard MSDN COM property. This property indicates that the property appears in an object browser (which does not show property values), but does not appear in a properties browser (which does show property values).

(Default = Cleared)

requestedit

The requestedit property is a standard MSDN COM property. This property indicates that the property supports the OnRequestEdit notification.

(Default = Cleared)

restricted

The restricted property is a standard MSDN COM property. This property prevents the item from being used by a macro programmer.

(Default = Cleared)

string

The string property is a standard MSDN COM property. This property specifies a string.

(Default = empty string)

uidefault

The uidefault property is a standard MSDN COM property. This property returns or sets the UIDefault attribute of a member object. The UIDefault attribute indicates that the type information member is the default member for display in the user interface.

(Default = Cleared)

Class

The Class metaclass contains properties that control how classes are mapped to code when using COM.

In

The In property specifies how code is generated when the type is used with an argument that has the modifier In. In the IDL, the class is: [in] class>* In C++, it is realized as: class>*

(Default = $type)*

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier InOut. In the IDL, the class is:

[in,out] class>** In C++, it is realized as: class>**

*(Default = $type**)*

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier Out. In the IDL, the class is: [out] class>** In C++, it is realized as: class>**

*(Default = $type**)*

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type. In the IDL, the class is HRESULT.

(Default = $type)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return." (Default = $type)*

coclass

The coclass metaclass contains properties that affect COM coclasses.

AppendToClause

The AppendToClause property is an optional verbatim property that enables you to add free text to be generated into the end of the description clause, before the closing bracket.

(Default = empty string)

appobject

The appobject property is a standard MSDN COM property. This property identifies the application object.

(Default = Cleared)

control

The control property is a standard MSDN COM property. This property indicates that the item represents a control from which a container site will derive additional type libraries or coclasses.

(Default = Cleared)

DefaultInterface

The DefaultInterface property specifies the default interface that a «COM Coclass» class should expose. This property is blank by default. To override the property, assign it the name of the «COM Interface» class that you want a coclass to expose. If you set the DefaultInterface property while a package or component is selected, the property is automatically applied to all new «COM Coclass» classes that you create in that package or component.

(Default = empty string)

DefaultSourceInterface

The DefaultSourceInterface property specifies the default source interface for a coclass.

(Default = empty string)

helpcontext

The helpcontext property is a standard MSDN COM property. This property sets the context in the Help file.

(Default = empty string)

helpstring

The helpstring property is a standard MSDN COM property. This property specifies the text that describes the element to which it applies, such as a library, dispinterface, or coclass.

(Default = empty string)

hidden

The hidden property is a standard MSDN COM property. This property indicates that the item exists, but should not be displayed in a user-oriented browser.

(Default = Cleared)

licensed

The licensed property is a standard MSDN COM property. This property indicates that the class is licensed.

(Default = Cleared)

noncreatable

The noncreatable property is a standard MSDN COM property. This property indicates that the class does not support creation at the top level (for example, through ITypeInfo::CreateInstance or CoCreateInstance). An object of such a class is usually obtained through a method call on another object.

(Default = Cleared)

uuid

The uuid property is a universal, unique identifier for a COM coclass (and other metaclasses). If this property is unspecified, Rational Rhapsody generates the value automatically.

(Default = empty string)

Configuration

The Configuration metaclass contains properties that affect the COM configuration.

COMClientStandardHeaders

The COMClientStandardHeaders property adds standard CPP includes for COM clients.

(Default = "comdef.h")

COMEnable

The COMEnable property specifies whether Rational Rhapsody should generate COM client code. The possible values are as follows:

- Client - Generate COM client code.
- No - Do not generate COM client code.

(Default = No)

COMInitialize

The COMInitialize property adds COM initialization code to the client's main() section.

(Default = CoInitialize(NULL);)

COMUnInitialize

The COMUnInitialize property adds COM uninitialization code to the client's main() section.

(Default = CoUninitialize(NULL);)

CreateInitialInstance

The CreateInitialInstance property is a template that creates an initial instance of a COM coclass for a COM client. The «COM TLB» stereotype is applied to components that are to be built into a TLB (a library of COM interfaces), and a ProxyStub.dll file. Running a make on such components runs first the Microsoft MIDL compiler and then the C++ compiler. The first (MIDL) phase of the make yields a TLB file. It also yields the sources for the ProxyStub.dll file, if the GenerateProxyStubDll property is set to True. In this case, the second (C++ compiler) phase compiles the sources yielded by the first phase into the ProxyStub.dll file. If the property is False, the ProxyStub.dll file is not built. A component with a «COM TLB» stereotype can have within its scope only the following elements:

- Packages stereotyped as «COM Library»
- Classes stereotyped as «COM Interface»
- Classes stereotyped as «COM Coclass»

The default value is as follows: IUnknownPtr m_pUnk$class(__uuidof($class), NULL,CLSCTX_ALL))

DestroyInitialInstance

The DestroyInitialInstance property is a template for destroying the initial instance of a COM coclass.

(Default = m_pUnk$class = NULL;)

GenerateProxyStubDll

The GenerateProxyStubDll property specifies whether the ProxyStub.dll file is built. See the

CreateInitialInstance property for more information.

(Default = Cleared)

MIDLCommand

The MIDLCommand property specifies the MIDL compiler command. The default value is as follows:

/* midl /tlb "$MIDLOutTypeLib" /h "$MIDLOutHeader" /iid "$MIDLOutIDFileName" /Oicf */

ProxyStubCommand

The ProxyStubCommand property specifies the makefile command to create a ProxyStub DLL. The default value is as follows:

/* $ProxyStubDllName: dlldata.obj $ComLibraryPackage_p.obj $ComLibraryPackage_i.obj link /dll /out:$ProxyStubDllName / def:$ProxyStubDefFileName /entry:DllMain dlldata.obj $ComLibraryPackage_p.obj $ComLibraryPackage_i.obj \ kernel32.lib rpcndr.lib rpcns4.lib rpcrt4.lib oleaut32.lib uuid.lib .c.obj: cl /c /Ox /DWIN32 /D_WIN32_WINNT=0x0400 / DREGISTER_PROXY_DLL $ */

ProxyStubDefFileName

The ProxyStubDefFileName property specifies the name of the .def file used to generate the ProxyStub. The default is $componentps.def, where $component is replaced with the name of the server.

(Default = $componentps.def)

ProxyStubDllName

The ProxyStubDllName property sets the name of the generated ProxyStub DLL.

(Default = $componentps.dll)

IDL

The IDL metaclass contains properties that affect the COM IDL™.

IDL_StandardImport

The IDL_StandardImport property imports the user-specified IDL file into all generated IDL files. The default value is as follows:

/* import "oaidl.idl"; import "ocidl.idl"; */

IDL_StandardInclude

The IDL_StandardInclude property includes a specified IDL file into all generated IDL files.

(Default = empty string)

IDLExtension

The IDLExtension property specifies the default extension for the IDL file.

(Default = .idl)

IDLImportFormat

The IDLImportFormat property sets the format of an imported IDL file.

(Default = import "$FILENAME";)

IDLIncludeFormat

The IDLIncludeFormat property sets the format of an included IDL file.

(Default = #include "$FILENAME")

InterfaceDeclaration

The InterfaceDeclaration property is a template that specifies how the COM interface should be declared in the IDL file.

(Default = interface $interface;)

MIDLOutHeader

The MIDLOutHeader property sets the name of the header file generated by the MIDL compiler.

(Default = $Package.h)

MIDLOutIDFileName

The MIDLOutIDFileName property sets the name of the ID file generated by the MIDL compiler.

(Default = $Package.i.c)

MIDLOutTypeLib

The MIDLOutTypeLib property sets the name of the type library file generated by the MIDL compiler.

(Default = $Package$TypeLibExtension)

TypeLib_StandardImport

The TypeLib_StandardImport property specifies the standard import files for a type library file. The default value is as follows:

importlib("stdole32.tlb"); importlib("stdole2.tlb");

TypeLibExtension

The TypeLibExtension property specifies the extension for a type library file.

(Default = .tlb)

TypeLibImportFormat

The TypeLibImportFormat property generates a .tlb extension for a file. You can add import statements manually by setting the TypeLibImportFormat property. The default value is as follows:

importlib("$FILENAME");

Interface

The Interface metaclass contains properties that affect the COM interface.

AppendToClause

The AppendToClause property is an optional verbatim property that enables you to add free text to be generated into the end of the description clause, before the closing bracket.

(Default = empty string)

ExternalInclude

The ExternalInclude property adds all external includes in the COM interface and COM library.

(Default = empty MultiLine)

helpcontext

The helpcontext property is a standard MSDN COM property. This property sets the context in the Help file.

(Default = empty string)

helpstring

The helpstring property is a standard MSDN COM property. This property specifies the text that describes the element to which it applies, such as a library, dispinterface, or coclass.

(Default = empty string)

hidden

The hidden property is a standard MSDN COM property. This property indicates that the item exists, but should not be displayed in a user-oriented browser.

(Default = Cleared)

local

The local property is a standard MSDN COM property. This property specifies to the MIDL compiler that an interface or a function is not remote.

(Default = Cleared)

nonextensible

The nonextensible property is a standard MSDN COM property. This property indicates that the IDispatch implementation includes only the properties and methods listed in the interface description.

(Default = Cleared)

object

The object property is a standard MSDN COM property. This property identifies a COM interface.

(Default = Checked)

oleautomation

The oleautomation property is a standard MSDN COM property. This property indicates that an interface is compatible with Automation.

(Default = Cleared)

pointer_default

The pointer_default property is a standard MSDN COM property. This property specifies the default pointer attribute for all pointers except top-level pointers that appear in parameter lists. The possible values are as follows:

- unique - Specifies a unique pointer
- ptr - Specifies a full pointer
- ref - Specifies a reference pointer

(Default = unique)

replaceable

The replaceable property is a standard MSDN COM property.

(Default = Cleared)

Type

The uuid property specifies the type of the COM interface. The possible values are as follows:

- Dual - An interface that exposes properties and methods through IDispatch and directly through the VTBL.
- Custom - A user-defined interface
- dispinterface - An interface that defines a set of properties and methods on which you can call IDispatch::Invoke

(Default = Dual)

uuid

The uuid property is a universal, unique identifier for a COM coclass (and other metaclasses). If this property is unspecified, Rational Rhapsody generates the value automatically.

(Default = empty string)

Library

The Library metaclass contains properties that affect COM libraries.

AppendToClause

The AppendToClause property is an optional verbatim property that enables you to add free text to be generated into the end of the description clause, before the closing bracket.

(Default = empty string)

control

The control property is a standard MSDN COM property. This property indicates that the item represents a control from which a container site will derive additional type libraries or coclasses.

(Default = Cleared)

ExternalInclude

The ExternalInclude property adds all external includes in the COM interface and COM library.

(Default = empty MultiLine)

helpcontext

The helpcontext property is a standard MSDN COM property. This property sets the context in the Help file. (Default = empty string)

helpfile

The helpfile property is a standard MSDN COM property. This property sets the name of the Help file.

(Default = empty string)

helpstring

The helpstring property is a standard MSDN COM property. This property specifies the text that describes the element to which it applies, such as a library, dispinterface, or coclass.

(Default = empty string)

helpstringdll

The helpstringdll property is a standard MSDN COM property. This property sets the name of the DLL to use to perform the document string lookup (localization).

(Default = empty string)

hidden

The hidden property is a standard MSDN COM property. This property indicates that the item exists, but should not be displayed in a user-oriented browser.

(Default = Cleared)

IncludePath

The IncludePath property specifies the COM library's path.

(Default = empty string)

lcid

The lcid property is a standard MSDN COM property. This property indicates that the parameter is a locale ID (LCID).

(Default = empty string)

restricted

The restricted property is a standard MSDN COM property. This property prevents the item from being used by a macro programmer.

(Default = Cleared)

uuid

The uuid property is a universal, unique identifier for a COM coclass (and other metaclasses). If this property is unspecified, Rational Rhapsody generates the value automatically.

(Default = empty string)

version

The version property specifies the version number of the COM Type Library.

(Default = 1.0)

Operation

The Operation metaclass contains properties that affect COM operations.

AppendToClause

The AppendToClause property is an optional verbatim property that enables you to add free text to be generated into the end of the description clause, before the closing bracket.

(Default = empty string)

call_as

The call_as property is a standard MSDN COM property. This property enables mapping a function, which cannot be called remotely, to a remote function.

(Default = empty string)

callback

The callback property is a standard MSDN COM property. This property declares a static callback function that exists on the client side of the distributed application. Callback functions provide a way for the server to execute code on the client.

(Default = Cleared)

helpcontext

The helpcontext property is a standard MSDN COM property. This property sets the context in the Help file.

(Default = empty string)

helpstring

The helpstring property is a standard MSDN COM property. This property specifies the text that describes the element to which it applies, such as a library, dispinterface, or coclass.

(Default = empty string)

hidden

The hidden property is a standard MSDN COM property. This property indicates that the item exists, but should not be displayed in a user-oriented browser.

(Default = Cleared)

id

The id property specifies the ID used for get/put methods written to the COM interface file. Relations between COM interfaces are treated like COM interface attributes, and the required get/put methods are written to the COM interface file.

(Default = empty string)

local

The local property is a standard MSDN COM property. This property specifies to the MIDL compiler that an interface or a function is not remote.

(Default = Cleared)

restricted

The restricted property is a standard MSDN COM property. This property prevents the item from being used by a macro programmer.

(Default = Cleared)

source

The source property is a standard MSDN COM property. This property indicates that a member of a coclass, property, or method is a source of events. For a member of a coclass, this attribute means that the member is called rather than implemented.

(Default = Cleared)

vararg

The vararg property is a standard MSDN COM property. This property specifies that the function takes a variable number of parameters. To accomplish this, the last parameter must be a safe array of VARIANT type that contains all the remaining parameters.

(Default = Cleared)

Relation

The Relation metaclass contains a property that affects COM relations.

id

The id property specifies the ID used for get/put methods written to the COM interface file. Relations between COM interfaces are treated like COM interface attributes, and the required get/put methods are written to the COM interface file.

(Default = empty string)

ComponentDiagram

The ComponentDiagram subject contains the following metaclasses:

- AutoPopulate
- Class
- Complete
- Component
- ComponentDiagramGE
- Depends
- FileComponent
- Flow
- FolderComponent
- CompRealization

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Bottom-Top

Class

The Class metaclass contains properties that control how classes are displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 0,255,255

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

Default =

name_color

The name_color property specifies the default color of names of graphical items.

Default =

ShowAttributes

The ShowAttributes property specifies which attributes are shown in an object box in a component diagram. The possible values are as follows:

- All - Show all attributes.
- None - Do not show any attributes.
- Public - Show only the public attributes.
- Explicit - Show only those attributes that you have explicitly selected.

Default = None

ShowBitmapMarkers

The ShowBitmapMarkers property specifies whether to show bitmap markers for stereotypes.

Default = True

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are

as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowOperations

The ShowOperations property specifies which operations to show in an object box in a component or object model diagram. The possible values are as follows:

- All - Show all operations.
- None - Do not show any operations.
- Public - Show only the public operations.
- Explicit - Show only those operations that you have explicitly selected.

Default = None

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Comment

The Comment metaclass contains properties that control the appearance of comments in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

Component

The Component metaclass contains properties that control how components are displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action

state lines).

Default =

name_color

The name_color property specifies the default color of names of graphical items.

Default =

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

ComponentDiagramGE

The ComponentDiagramGE metaclass contains a property that controls the fill color of component diagrams.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

Constraint

The Constraint metaclass contains properties that control the appearance of constraints in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Depends

The Depends metaclass contains a property that controls the appearance of dependency relation lines in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

Default = 1

name_color

The name_color property specifies the default color of names of graphical items.

Default =

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

FileComponent

The FileComponent metaclass contains properties that control how file components are displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

Default =

name_color

The name_color property specifies the default color of names of graphical items.

Default =

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Flow

The Flow metaclass contains properties that control how information flows are displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

flowKeyword

The flowKeyword property is a Boolean value that specifies whether the flow keyword for the information flow is displayed in the diagram.

Default = Checked

infoItemsColor

The infoItemsColor property specifies the color used to draw information items in diagrams.

Default =

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

Default =

FolderComponent

The FolderComponent metaclass contains properties that control how folder components are displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

Default =

name_color

The name_color property specifies the default color of names of graphical items.

Default =

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

InterfaceComponent

The InterfaceComponent metaclass contains properties that control how interface components are displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 255,255,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 255,255,0)

Note

The Note metaclass contains properties that control the appearance of notes in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,128,64)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,128,255)

CompRealization

The CompRealization metaclass contains a property that controls how realization (instantiation) is displayed in component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

Default =

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines).

Default =

name_color

The name_color property specifies the default color of names of graphical items.

Default =

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

Requirement

The Requirement metaclass contains properties that control the appearance of requirements component diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

ConfigurationManagement

The ConfigurationManagement properties specify values and command strings needed by various configuration management (CM) tools to interface with Rational Rhapsody.

It contains the following metaclasses:

- ClearCase - The ClearCase metaclass contains properties that enable your ClearCase implementation with Rational Rhapsody.
- General - This metaclass informs Rhapsody which CM tool you are using, and the length of that particular tool's timeout for commands.
- PVCS - The PVCS metaclass contains properties that enable you to customize PVCS Dimensions.
- SCC - The SCC metaclass contains properties that enable you to use the SCC interface with Rational Rhapsody.
- SourceIntegrity - The SourceIntegrity metaclass contains properties that enable you to use the SourceIntegrity inplementation with Rational Rhapsody.
- Synergy - These properties control the interaction of Rational Rhapsody with the SYNERGY configuration management system.

ClearCase

The ClearCase metaclass contains properties that enable you to customize your CM tool.

AddMember

The AddMember property specifies the command used to add an item to the archive.

For example, using ClearCase, this command would be as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc $rhpdirectory ; cleartool mkelem -eltype text_file -nc $unit; cleartool checkin -nc $rhpdirectory"

In this case, the argument to the command to run the Executer ($OMROOT/etc/Executer.exe) consists of a list of three executable commands: cleartool checkout -reserved -nc $rhpdirectory cleartool mkelem -eltype text_file -nc $unit cleartool checkin -nc $rhpdirectory

The first command, cleartool checkout -reserved -nc, is defined within ClearCase to check out an item. In this case, the item is checked out from the _rpy directory (as indicated by the variable $rhpdirectory).

In ClearCase, to check an item out of the archive means to create a view-private, modifiable copy of a version. The option -reserved checks the item out as locked (R/W for the owner). ClearCase expects items to be checked out of the user's repository before they are checked into the archive.

The option -nc (no additional comment) is defined in ClearCase to create an event record with no user-supplied comment string.

The second command, cleartool mkelem -eltype text_file -nc $unit, creates an element of type text_file and assigns this element to the variable $unit, which represents the unit of collaboration.

The third command, cleartool checkin -nc $rhpdirectory, checks the unit in the _rpy directory into the archive.

Because a second argument is not provided, the Executer runs these commands from the current (_rpy) directory.

The default is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$UnitDirPath" ; cleartool mkelem -eltype text_file -nc "$UnitPath" ; cleartool checkin -nc "$UnitDirPath""

AddMember_ControlledFile

For ClearCase only and like AddMember, this property is used for adding items to archive. However, the string contained in this property does not include the argument -eltype text.

The removal of this argument allows ClearCase to stores binary files, as well, and this is necessary for storing Controlled Files.

When Controlled Files are stored in ClearCase, the value of this property is used. When other files are stored, the value contained in AddMember is used.

The default is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$UnitDirPath" ; cleartool mkelem -nc "$UnitPath" ; cleartool checkin -nc "$UnitDirPath""

AddMember_WithoutCheckOutCheckInDirectory

This property allows an item to be added to the archive without having to check out the parent directory, and check it back in after the item has been added. The use of this property allows you to check out and check in the parent directory separately.

Default = cleartool mkelem -eltype text_file -nc "$UnitPath"

AddToArchiveAfterCreateUnitActivation

When you create an element in a Rhapsody project that is not a unit by default (an actor, or class, for example), you can opt to create a unit from that element. When you do so, and this property is enabled, the new unit is automatically added as a unit in ClearCase.

The possible values are as follows:

- Disabled - No units are archived
- UserConfirmation - prompts the user for confirmation before archiving the unit.

- Automatic - automatically archives the unit without asking the user.

Default = Disable

Archive

The Archive property specifies the archive file for a project. This property is implicitly set when you use the Connect to Archive option within Rational Rhapsody, which uses the command string specified by the ConnectToCMRepository property.

Do not set the Archive property manually.

ArchiveRoot

The ArchiveRoot property is reserved for future use.

ArchiveSelection

The ArchiveSelection property specifies whether the archive façade is a file or directory. Most CM tools expect a file, whereas others expect a directory.

The Browse button in the Connect to Archive window runs a different browse utility for file-based and directory-based archives. When the ArchiveSelection property is set to File, the Open dialog for files is displayed by default. In this case, you cannot select a directory as the archive. If the archive is a directory rather than a file, set the ArchiveSelection property to Directory. The Browse for Folder window is displayed instead, which allows you to select a directory.

The possible values are as follows:

- Directory - The CM archive is a directory.
- File - The CM archive is a file. This is the default for PVCS.
- None - Neither a file nor directory is expected.

With ClearCase, the ArchiveSelection property is not set (blank). This should not be changed. ClearCase ignores the path to the archive, so the ArchiveSelection property is irrelevant. Similarly, the Browse button in the Connect to Archive dialog is disabled for ClearCase.

AskOnCheckoutReadWrite

The AskOnCheckoutReadWrite property is a Boolean value that specifies whether Rational Rhapsody should prompt users if they want to perform a check out (because they already have a read/write copy of the file).

Default = Yes

AskOnLoadFromArchive

The AskOnLoadFromArchive property specifies whether to display a message when you check a file out of an archive that is not part of the current model.

If this property is set to Yes, Rational Rhapsody displays a window when you perform a checkout from the archive (or fetch operation for SCC) from either the ListArchive or Synchronize window.

When you check out a file from the archive and it is not the part of the current model, there are two possibilities:

- The corresponding parent unit is not loaded in the model.
- The parent unit, if loaded, is not the latest one.

Therefore, before Rhapsody checks out an element from an archive, it checks whether the file is already part of the model. If not, it checks whether there is a stub unit in the model that refers to the same file. If neither criteria is met, Rational Rhapsody displays an informational message.

Default = Yes

AuxProjPath

The AuxProjPath property is a string that identifies the SCC project path. Do not change this property. Removing the property value will disconnect the link between the Rational Rhapsody project and the SCC project.

If you remove this value, you must also remove the ProjName property.

Default = empty string

BaseAwareAutoMergeInvocation

The BaseAwareAutoMergeInvocation property specifies how to run the external textual DiffMerge tool which supports a base-aware detection of triviality of textual difference and a base-aware automatic merging between a base unit file and two other unit files. An automatic model merge operation can be performed only if all the differences in the current diff session are trivial differences.

If two units are being compared with a base unit, this three-way comparison makes it possible for the DiffMerge tool to determine automatically the need for some merges using the concept of trivial versus non-trivial differences. For a difference in which only one unit differs from the base unit, it is identified as a non-conflicting difference or trivial difference. Similarly, if both of the units are different from base contributor but the differences are same, then it is also a trivial difference.

This applies to differences between model elements or between the attributes of model elements. However, if one unit contains a difference that does not appear in either the base unit or the other unit being compared, this is a non-trivial difference that must be resolved by the developer.

During the automatic merge, all of the differences are automatically accepted.

Default = cleardiff -out $output -base $sourceBase -abo -qui $source1 $source2

BaseAwareDiffInvocation

The BaseAwareAutoMergeInvocation property specifies how to run an external textual DiffMerge tool supporting a base-aware comparison and merging in Base Aware Diff mode between a base unit file and two other unit files. Possible tools supporting a base-aware comparison include TkDiff and the ClearCase textual DiffMerge tools ClearDiff and ClearDiffMrg. $source1: First unit selected in the window. $source2: Second unit selected in the window. $sourceBase: The text file containing compared values from the base. Default = cleardiffmrg -base $sourceBase $source1 $source2

BaseAwareDiffMergeInvocation

The BaseAwareDiffMergeInvocation property specifies how to run the external textual DiffMerge tool supporting a base-aware comparison and merging in Base Aware Merge mode between a base unit file and two other unit files. Default = $BaseAwareDiffInvocation -out $output

BaseAwareTextDiffMergeEnabled

Determines whether a base-aware (three-unit) textual DiffMerge tool is available to be started. Default = Checked

CallCheckOutOnSynchronize

The CallCheckOutOnSynchronize property is a Boolean value that determines whether a checkout operation when you specify synchronization in Rational Rhapsody. Some CM tools require a full checkout of the selected elements, followed by an Add to Model operation. However, ClearCase does not require a checkout. Put a Check in this property check box to ensure a checkout operation.

Default = Cleared

CheckIn

The CheckIn property specifies the command used to check an item into the archive using the main Configuration Items window.

This command is specific to the CM tool in use.

This command references the LogPart property, which in turn references the internal variable $log, and the variable $unit.

Default = cleartool checkin $LogPart $UnitPath

By default, ClearCase does not allow you to check in an unmodified version of an item. Rhapsody includes a script, SensativeCheckin.bat, that checks whether a checked-out version of an item is different from the previous version, and then performs the checkin. If the item has not been changed, the script automatically performs an “uncheckout” operation for that item.

To use this batch file, set the ConfigurationManagement::ClearCase::CheckIn property to $OMROOT/etc/SensativeCheckin.bat $unit $log.

CheckInRevisionPart

The CheckInRevision property defines a new revision number used to check in a unit using the CheckIn property. The value for $CheckInRevisionPart is taken from the Label field in the dialog for checking in a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check In the selected items.

CheckOut

The CheckOut property specifies the command used to check an item out of the archive using the main Configuration Items dialog.

If the item is locked, the variable $mode is replaced by the contents of the ReadWrite property; otherwise, it is replaced by the contents of the ReadOnly property. LabelPart is also evaluated and replaced by the result.

The default value is as follows:

cleartool checkout -nc $mode $LabelPart

This command references the LabelPart property and the $mode.

CheckOutCheckInDirectoryOnceDuringAddToArchive

The CheckOutCheckInDirectoryOnceDuringAddToArchive property specifies that a directory is checked out only once when more than one file is added to the directory. The directory is checked in when all the files have been added.

Default = Cleared

CheckOutFromArchive

The CheckOutFromArchive property specifies the command used to check an item out of the archive directly using the List Archive window. For all the currently supported tools (ClearCase and PVCS Dimensions), this command is the same as that stored in the CheckOut property. Therefore, its value is $CheckOut.

Default = $CheckOut

CheckOutRevisionPart

The CheckOutRevisionPart property defines a revision number used to check out a unit using the CheckOut property. The value for $CheckOutRevisionPart is taken from the Revision/Label field in the

dialog for checking out a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check Out the selected items.

The default is ($label ? -r $label : -h).

CMHeaderItsLockedBy

The CMHeaderItsLockedBy property specifies the regular expression used to extract the name of the person who last locked the unit.

For all tools, an empty string in CMHeader* means that the unit does not contain this type of information, therefore it is missing from the Configuration Items dialog when you select the List Archive option.

All properties of the format CMHeader* are regular expressions that extract header information embedded into a configuration item when it is first checked into the archive. Embedded information can include such things as the ID of the item, its revision number, and the name of the person who last locked it (the “locker”).

Header information is contained in predefined, tool-specific keywords, which the various CM tools know how to expand.

The list of predefined keywords embedded in an item when it is first checked in is contained in a file stored in the HeaderFile property.

All regular expressions in CMHeader* properties match only one set of parentheses.

CMHeaderItsVersion

The CMHeaderItsVersion property specifies the regular expression that extracts the version information.

Consider the following PVCS expression: \$Revision: +([0-9\.]+)

This expression searches the header for a string that begins with \$Revision: and contains a version number that can consist of one or more digits 0 through 9, the backslash character (\), or a period.

All properties of the format CMHeader* are regular expressions that extract header information embedded into a configuration item when it is first checked into the archive. Embedded information can include such things as the ID of the item, its revision number, and the name of the person who last locked it (the “locker”).

Header information is contained in predefined, tool-specific keywords, which the various CM tools know how to expand.

The list of predefined keywords embedded in an item when it is first checked in is contained in a file stored in the HeaderFile property.

All regular expressions in CMHeader* properties match only one set of parentheses.

CommentsRequiredForCheckIn

The CommentsRequiredForCheckIn property sets whether comments are required upon SCC Check In.

When this flag is set to Checked, comments are required for successful SCC Check In operation.

Default = Cleared

ConnectToCMRepository

The ConnectToCMRepository property specifies the command used to connect Rhapsody to a CM archive. For some tools, this is simply an echo.

For ClearCase, the connect command is as follows: "$OMROOT/etc/Executer.exe" "move $rhpdirectory $rhpdirectory.orig ; cleartool mkelem -eltype directory -nc -nco $rhpdirectory ;" ".."

This is a command to run the Rational Rhapsody Executer with two arguments:

- The first argument consists of two executable commands: move $rhpdirectory $rhpdirectory.orig cleartool mkelem -eltype directory -nc -nco $rhpdirectory
- The first command backs up the repository (the user's _rpy directory). The second command is defined within ClearCase to create an element of type directory. The -nc option (no additional comment) creates an event record with no user-supplied comment string. The new directory points to the repository.
- The second argument, "..", tells the Executer to run the commands from the directory just above the current one.

A side-effect of the ConnectToCMRepository property is that it sets the Archive property to the location of the CM archive, even if a CM command is not actually executed.

Delete

The Delete property specifies the script that deletes a particular item from the current ClearCase directory element.

When you delete a Rhapsody unit and the project settings indicate that the CM tool is ClearCase, Rational Rhapsody runs this delete script to remove the deleted item from the ClearCase view as well.

If the property does not exist, or if is it empty, the unit is removed from the Rational Rhapsody model, but not from the ClearCase view.

By default, this feature is disabled. To enable it, set the property DeleteActivation.

The default value of the Delete property is as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$UnitDirPath" ; cleartool rmname -nc "$UnitPath" ; cleartool checkin -nc "$UnitDirPath""

It uses the following keywords:

- $units - Specifies the full path names of the unit file names, separated by spaces

- $dirs - Specifies the names of the unit directories, separated by spaces

DeleteActivation

The DeleteActivation property is a Boolean value that specifies whether deletion of units from the Rational Rhapsody model triggers a delete command in the archive.

The possible values are as follows:

- Disable - Disable the trigger.
- UserConfirmation - Prompt the user for confirmation before performing the deletion.
- Automatic - Automatically trigger the delete command in the archive when a unit is deleted in Rational Rhapsody.

Default = Disable

DeleteDirectory

The DeleteDirectory property specifies the command to delete a directory in the ClearCase configuration management system.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$UnitDirPath\.." ; cleartool rmname -nc "$UnitDirPath" ; cleartool checkin -nc "$UnitDirPath\.."

DiffInvocation

The DiffInvocation property specifies the command to run the external, textual DiffMerge tool. For ClearCase, this property runs Clearcase's diff tool.

Rhapsody searches for the property value as follows:

- First, it searches through the CM property values.
- If the property is not found or if it is set to empty, Rational Rhapsody searches under General::DiffMerge.

By default, site-specific properties override the factory properties.

Default = cleardiffmrg $source1 $source2

DiffMergeInvocation

The DiffMergeInvocation property specifies the command to run the external, textual Diff/Merge tool.

For PVCS and Clearcase, this property runs the corresponding CM tool's diff tool. For the other CM tools, this property is set to an empty string.

Rhapsody searches for the property value as follows:

- First, it searches through the CM property values.
- If the property is not found or if it is set to empty, Rational Rhapsody searches under General::DiffMerge.

By default, site-specific properties override the factory properties.

Default = $DiffInvocation -out $output

EnableSCCCancel

The boolean property EnableSCCCancel is used to provide a cancel option during CM operations.

When set to True, the SCC provider displays its cancel window during CM operations.

The DeleteActivation property specifies whether deleting units from the Rational Rhapsody model will trigger the delete command in the archive.

The possible values are as follows:

- Cleared - The delete operation is disabled.
- UserConfirmation - The delete operation is performed only on user confirmation.
- Automatic - If you delete a unit from Rhapsody, the unit is automatically removed from the archive without any notification.

Default = Cleared

Fetch

The Fetch property specifies the command used by the Rational Rhapsody Diff/Merge to fetch files from the CM tool.

Default = $OMROOT/etc/copy.bat $FetchLabelPart $targetDir\$unit

FetchFromArchive

The FetchFromArchive property specifies the command used by the Rational Rhapsody Diff/Merge to fetch files from the CM archive.

Default = $Fetch

FetchLabelPart

The FetchLabelPart property is a string used by the Fetch property to identify units by label. For more information, see the Fetch property listed previously in this metaclass.

The default is ($label ? $UnitPath@@$label : $UnitPath).

FooterFile

The FooterFile property specifies the file footer.

HeaderFile

The HeaderFile property specifies the file header. This property is reserved for future use.

History

The History property specifies the batch script that enables you to view the version tree of a given item.

The default value is as follows: cleartool lsvtree -graph $UnitPath

InValidCharactersInRevisionDescription

The InValidCharactersInRevisionDescription property provides a list of iInvalid characters in the Revision Description (or comments) during a Check In operation. The list of invalid characters - provided by the user - validates the revision description and notifies the user if there are any invalid characters. By default this property value is set to ">" and users can append more characters to it. This is property is used only in ClearCase Batch Mode CheckIn operation.

LabelPart

The LabelPart property specifies how to embed a revision label.

The syntax for embedding labels in ClearCase is: ($label ? -version $UnitPath@@$label : $UnitPath)

This expression uses the (Exp1 ? Exp2 : Exp3) construct. If you entered a label in the Revision/Label field in the Check In or Check Out window, $label is True and LabelPart evaluates to -version $unit@@$label, where $unit and $label are replaced by their respective values.

Otherwise, $label is False and LabelPart evaluates to $unit.

ListArchive

The ListArchive property specifies the command to list the contents of the archive.

For example, the command to list the archive in ClearCase is:

cleartool ls -vob_only -long -recurse

Expansion of ListArchive is done by “simple” substitution. The expanded command is executed as a shell command and the output is assigned to a temporary string inside Rhapsody. This string is then matched against the relevant regular expression found in ListArchive* to extract specific information from the output.

The following example shows sample output from the ListArchive command:

version animPingPong.cfg@@\main\2 Rule: element * \main\LATEST version animPingPongFlat.cfg@@\main\2 Rule: element * \main\LATEST version animPingPongMultiThread.cfg@@\main\2 Rule: element * \main\LATEST version Default.sbs@@\main\2 Rule: element * \main\LATEST version DefaultConfig.cfg@@\main\2 Rule: element * \main\LATEST version Model1.omd@@\main\4 Rule: element * \main\LATEST version MSC1.msc@@\main\3 Rule: element * \main\LATEST

An empty string in any of the ListArchive* properties means that the configuration item does not contain this information; therefore, it is missing from the archive listing.

ListArchiveItsLockedBy

The ListArchiveItsLockedBy property specifies the regular expression that extracts from the output of the List Archive command the name of the person who last locked a configuration item.

Consider the following PVCS expression: Locked by: +([0-9a-zA-Z]+)

This expression tells the interpreter to look in the header information for “Locked by:”string beginning. Once the string is matched, only the regular expression between the brackets is taken as the “locker.”

For example, the ClearCase CheckIn command is: cleartool checkin $LogPart $unit

This command references the LogPart property, which in turn references the internal variable $log, and the variable $unit.

ListArchiveItsVersion

The ListArchiveItsVersion property specifies the version of the CM archive.

The default value is as follows:

@@ ?\[?([0-9a-zA-Z_: *\-\.\\]+)\]?

ListArchiveItsWorkingFile

The ListArchiveItsWorkingFile specifies the regular expression that extracts from the output of the List Archive command the working file of an item.

The default is as follows: ([\\0-9\.a-zA-Z_,-]+)@@

As an example, this command "([0-9a-zA-Z_\.]+)@@" tells the interpreter that the working file is

indicated by the part of the output string preceding the two @ symbols. Once the string is matched, only the regular expression between the brackets is taken as the working file.

ListArchiveRevisionPart

The ListArchiveRevisionPart property is a string that represents the revision part from the output of the List Archive command.

LockItem

The LockItem property is a string that specifies the command tool used to lock an item in the archive.

LogPart

The LogPart property specifies how to embed a log, if provided, in a CM command. The log is the comment entered in the Revision/Description field in the Check In window.

For example, a log is embedded in ClearCase as follows: ($log ? -c $log : -nc)

This expression uses the (Exp1 : Exp2 : Exp3) construct. If you entered a comment in the Check In dialog, $log is True and LogPart evaluates to -c, followed by the comment string. Otherwise, $log is False and LogPart evaluates to -nc.

MakeCMShadowDir

The MakeCMShadowDir property specifies the ClearCase command to create a directory as a VOB element.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$parentdir" ; move "$fulldir" "$fulldir.orig" ; cleartool mkelem -eltype directory -nc -nco "$fulldir" ; cleartool checkin -nc "$parentdir" ; copy "$fulldir.orig" "$fulldir" ; " .."

MakeCMShadowDirActivation

The MakeCMShadowDirActivation property controls whether new directories created by a save in Rational Rhapsody is elements in ClearCase.

The possible values are as follows:

- Disable - Disable this functionality.
- UserConfirmation - Prompt the user for confirmation before creating the elements.
- Automatic - Automatically create elements whenever new directories are created by a save in Rational Rhapsody.

If you set this property to Automatic, every new package that is saved will create a new CM directory, including branches. If you do not want this to occur, set this property to UserConfirmation.

Default = Disable

MergeOutput

The MergeOutput property specifies the file that will hold the results of a merge operation.

Default = $temp\out.txt

ModePart

The ModePart property specifies the locking mode of a configuration item. This is defined as: $mode

If the item is locked, $mode is replaced by the value of the ReadWrite property; otherwise, it is replaced by the value of the ReadOnly property.

Move

The Move property specifies the ClearCase command for a unit move.

The default value is as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$olddir" "$newdir"; cleartool mv -nc "$oldName" "$newName" ; cleartool checkin -nc "$olddir" "$newdir" "

MoveActivation

The MoveActivation property is a Boolean value that specifies whether moving units in the Rational Rhapsody model (in a way that changes the unit file location on the hard drive) will trigger a rename command in the archive.

The possible values are as follows:

- Disable - Disable this functionality.
- UserConfirmation - Prompt the user for confirmation before renaming the elements in the archive.
- Automatic - Automatically rename elements whenever units are moved in the Rational Rhapsody model.

Default = Disable

MoveDirectory

The MoveDirectory property specifies the command to move a directory in the ClearCase configuration management system.

The default value is as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$olddir" "$newdir"; cleartool mv -nc "$oldName" "$newName" ; cleartool checkin -nc "$olddir" "$newdir" "

MultiRecordDelimiter

The MultiRecordDelimiter property specifies the regular expression representing the symbol used to separate multiple records in an archive listing.

For example, in ClearCase, the multiple-record delimiter is defined as:

(version)|(file element)

Multiple records are separated by the string version, located at the beginning of each line. See the ListArchive property for sample output.

OperationErrorPattern

The OperationErrorPattern property notifies you that the specified error occurred during batch mode. Rhapsody searches all CM operation output for the string specified in this property. Rhapsody first checks for errors, then for warnings.

Note the following:

- The string pattern is not case-sensitive.
- It can be a regular expression.

For example, on ClearCase, you could specify the following string: cleartool: Error:

Default = empty string

OperationWarningPattern

The OperationWarningPattern property notifies you that the specified warning occurred during batch mode. Rhapsody searches all CM operation output for the string specified in this property. Rhapsody first checks for errors, then for warnings.

Note the following:

- The string pattern is not case-sensitive.
- It can be a regular expression.

For example, on ClearCase, you could specify the following string: cleartool: Warning

Default = empty string

PostConnectToCMRepository

The PostConnectToCMRepository property is used for internal purposes only. Do not change the value of this property.

ProjName

The ProjName property is a string that identifies the SCC project name. Do not change this property. Removing the property value will disconnect the link between the Rational Rhapsody project and the SCC project.

If you remove this value, you must also remove the AuxProjPath property.

Default = empty string

ReadOnly

The ReadOnly property specifies how to embed a ReadOnly flag in the CM command.

Default = -unreserved

ReadWrite

The ReadWrite property specifies how to embed a ReadWrite flag in the CM command.

For example, the ClearCase CheckIn command is: cleartool checkin $LogPart $unit

Default = -reserved

RedirectOutputToRhapsody

The UseSCCtool property is a Boolean value that specifies whether the resulting output of an SCC CM command should be redirected (displayed) in the Rational Rhapsody CM output window tab.

Default = Checked

Rename

The Rename property specifies the script that renames a particular item in the current ClearCase directory element.

When you rename a Rhapsody unit (either explicitly by changing the file name field in the Edit Unit window, or implicitly by changing the unit name) and the project settings indicate that the CM tool is ClearCase, Rational Rhapsody runs this rename script to rename the item in the ClearCase view as well.

If the property does not exist, or if is it empty, the unit is renamed in the Rational Rhapsody model, but not in the ClearCase view.

By default, this feature is disabled. To enable it, set the RenameActivation property.

The default value of the Rename property is as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc $dir ; cleartool mv -nc $oldName $newName ; cleartool checkin -nc $dir"

It uses the following keywords:

- $oldName - Specifies the full path name of the existing unit file name
- $newName - Specifies the full path name of the new unit file name
- $dir - Specifies the name of the unit directory

RenameActivation

The RenameActivation property is a Boolean value that specifies whether renaming units in the Rational Rhapsody model triggers a rename command in the archive.

The possible values are as follows:

- Disable - Disable the Rename functionality.
- UserConfirmation - Prompt the user for confirmation before renaming the elements from the archive.
- Automatic - Automatically rename the element units that are renamed in the Rational Rhapsody model.

Default = Disable

RenameDirectory

The RenameDirectory property specifies the command to rename a directory in the ClearCase configuration management system.

The default value is as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc $dir ; cleartool mv -nc $oldName $newName ; cleartool checkin -nc $dir"

The RenameActivation property controls whether the rename operation (specified by the Rename property) is enabled.

ReplaceNewLinesInRevisionDescriptionWithSpaces

When SensitiveCheckin.bat is used for check in, you can enable this property to properly format revision descriptions or comments that contain multiple lines of text.

Default = Cleared

Repository

The Repository property is used for internal purposes only. Do not change the value of this property.

The default is .\$SubDirs\$FileName.

For example, the ClearCase CheckIn command is:

cleartool checkin $LogPart $unit

This command references the LogPart property, which in turn references the internal variable $log and the variable $unit.

SaveOnCheckOut

The SaveOnCheckOut property is a Boolean value that specifies whether a Rhapsody save should be triggered whenever a checkout occurs.

Default = Checked

ShowNewItemsInSynchronize

The ShowNewItemsInSynchronize property is a Boolean value that is directly related to what you see in the Synchronize window.

If this property is set to No, new items that are added (by another member of the team) to the archive after the Rational Rhapsody project is open are not displayed.

Default = Yes

StoreInSeparateDirectoryActivation

The StoreInSeparateDirectoryActivation property affects how an existing package is converted. The following conditions apply with this property:

- Enable this property by selecting UserConfirmation or Automatic from the drop-down list. When an existing flat package is converted to a package as a directory, the directory is created on the configuration management side and the children of this package are moved to this directory.
- Disable this property by selecting Disable. When an existing flat package is converted to a package as a directory, the directory is removed on the configuration management side and the children of this package are removed as well.

Default = Disable

SupportTreeRepository

The SupportTreeRepository property is provided for backward compatibility to previous versions of Rational Rhapsody.

For PVCS Dimensions, this property is for internal purposes only. Do not change the value of this property.

If you checked in Rational Rhapsody projects to an SCC archive prior to Version 4.0, you must perform

the following steps:

- Create a directory in the CM tool with the same name of the directory that holds the .rpy file.
- Disconnect from the existing archive.
- Change the value of this property to an empty string.
- Reconnect to the archive.

The default value for PVCS is No; the default for SCC is an empty string.

UnLockItem

The UnLockItem property is a string that specifies the command used to release a lock placed on an item in the archive.

For example, the ClearCase CheckIn command is:

cleartool checkin $LogPart $unit

This command references the LogPart property, which in turn references the internal variable $log, and the variable $unit.

This command references the properties CheckInRevisionPart, ModePart, and LogPart and the internal variables $archive, $archivedirectory, and $unit.

General

The General properties inform Rhapsody which CM tool you are using, and the length of that particular tool’s timeout for commands.

CMConflictResolution

The CMConflictResolution property is related to CM synchronization and Rhapsody model changes.

When CM synchronization is enabled and you try to make changes in a Rhapsody model (delete, rename, or move), and Rhapsody finds that those changes cannot be made in the CM system, it displays an informative window (“Change in Directory Structure”).

The possible values are as follows:

- AskUser - Always ask the user.
- ModelOnly - Changes is done in the model, but the file and directory layout will remain the same.
- ModelAndFileSystem - Changes is done in both the model and the file/directory layout.

Default = AskUser

CMOperationEndSeparator

The CMOperationStartSeparator property is a MultiLine value that specifies the separator for the end of a CM operation. The text specified in this property is printed to the configuration management window after the CM operation has executed.

This property supports the following keywords:

- $Time - The current time
- $Date - The date
- $User - The current user name
- $Operation - The name of the operation.

Note that the CM output window is not cleared between consecutive CM operations. To clear the output window, right-click on the window. After you close a project, this window is cleared automatically.

Default = empty MultiLine

CMOperationStartSeparator

The CMOperationStartSeparator property is a MultiLine value that specifies the separator for the start of a CM operation. The text specified in this property is printed to the CM window as the header before the CM operation is executed.

This property supports the following keywords:

- $Time - The current time
- $Date - The date
- $User - The current user name
- $Operation - The name of the operation.

The default value is as follows:

==== $Operation; ==== Time: $Time; Date: $Date; User: $User ; ====

For example: ==== Checkout; Time: 2:38 Eastern Standard Time PM ; Date: Wed, 21, Nov 2001 ; User: npadmawar ; ====

Note that the CM output window is not cleared between consecutive CM operations. To clear the output window, right-click on the window. After you close a project, this window is cleared automatically.

CMTool

The CMTool property specifies which configuration management tool you are using. Valid properties for each CM tool are predefined in metaclasses of the same name. When evaluating property strings that reference other properties, Rational Rhapsody looks only within the same metaclass.

For example, the CheckOutFromArchive property for ClearCase references another property called CheckOut.

The possible values are:

- None
- ClearCase

Default = None

Note: If you are using SYNERGY as your CM tool, you need to set the UseSCCTool property to "Yes" and leave this property set to "None."

DefaultLockReserveOnCheckOut

The property DefaultLockReserveOnCheckOut provides a default lock or reserved value during a Checkout operation in Batchmode.

The default is Cleared, meaning that whenever a checkout operation is performed it is locked or reserved.

EncloseCommentsInQuotes

Comments provided during configuration management operations are enclosed in double quotes by default. These double quotes around comments cause problems in CM tools such as CM Synergy and ClearCase. To avoid double quotes in comments, the EncloseCommentsInQuotes property can be set to 'No' which prevents quotes from being enclosed around comments during CM operations. By default this property value is "Yes" which results in double quotes being enclosed around comments.

FilterUnresolvedUnitsInCMSynchDialog

This property allows you to filter the display of unresolved units in a CM synchronization window.

Default = Cleared. This means that unresolved units are displayed in a CM sychronization window.

GUI

The GUI property specifies the way units are displayed in the Configuration Items window.

The possible values are as follows:

- Flat - Units are displayed as a flat list.
- Tree - Units are displayed in a tree format:

It is possible to make multiple selections in the tree.

Selecting a higher-level unit in the tree does not automatically select subordinate units, unless you select the With Descendants option in the Check In/Out window.

If you select a unit that has subordinate units without selecting the With Descendants option, you might get stubs (unresolved units) for the subordinate units.

Default = Flat

ReportLoadingError

The ReportLoadingError property enables you to redirect loading errors.

After a checkout or fetch, Rational Rhapsody attempts to load the checked out element into the project. However, this operation might fail (for example, if the file is checked out to a wrong location, or is corrupt).

The possible values are as follows:

- OutputWindow - Display the error in the output window.
- MessageBox - Display the error in a message box.
- Both - Display the error in the output window and a message box.
- None - Do not display loading errors.

Default = OutputWindow

RunCMToolCommand

The RunCMToolCommand property specifies the command to execute the CM tool. This command is tied to a user-defined button.

Default = empty File

ToolCommandTimeOut

The ToolCommandTimeOut property specifies the time, in milliseconds, that Rational Rhapsody should wait for the CM tool to return its output before timing out.

This value should be higher for cross-network archives with a loaded network and for large units.

Default = 30000

UseHybridModeWhenPossible

The ClearCase SCC interface does not provide the required functionality to successfully perform certain CM operations, such as "Diff with Rational Rhapsody " and "Store in Separate Directory," which are provided in the Batch mode for ClearCase. In order to successfully perform these operations in ClearCase SCC mode, executing in the hybrid mode should be allowed. In this case, Rational Rhapsody will see if the SCC provider is ClearCase, and if so it will execute the corresponding batch commands.

Default = Checked

UserDefCommand_1

The UserDefCommand_1 property specifies the first parameter used in the command that is tied to a user-defined button for the CM tool.

Default = empty string

UserDefCommand_1_Title

In the Configuration Items window (File > Configuration Items) there are four user defined command buttons on the lower right side. The user can specify a title for each of these four command buttons (in this case UserDefCommand_1_Title). When rolled over with the pointer, the user-defined title is displayed.

Right-clicking on a unit in the browser and selecting Configuration Management from the submenu that appears displays CM operations and user-defined titles. When a user-defined title is selected, the command value is displayed on the status bar.

UserDefCommand_2

The UserDefCommand_2 property specifies the second parameter used in the command that is tied to a user-defined button for the CM tool.

Default = empty string

UserDefCommand_2_Title

In the Configuration Items window (File > Configuration Items) there are four user defined command buttons on the lower right side. The user can specify a title for each of these four command buttons (in this case UserDefCommand_2_Title). When rolled over with the pointer, the user-defined title is displayed.

Right-clicking on a unit in the browser and selecting Configuration Management from the submenu that appears displays CM operations and user-defined titles. When a user-defined title is selected, the command value is displayed on the status bar.

UserDefCommand_3

The UserDefCommand_3 property specifies the third parameter used in the command that is tied to a user-defined button for the CM tool. Default = empty string

UserDefCommand_3_Title

In the Configuration Items window (File > Configuration Items) there are four user defined command buttons on the lower right side. The user can specify a title for each of these four command buttons (in this

case UserDefCommand_3_Title). When rolled over with the pointer, the user-defined title is displayed.

Right-clicking on a unit in the browser and selecting Configuration Management from the submenu that appears displays CM operations and user-defined titles. When a user-defined title is selected, the command value is displayed on the status bar.

UserDefCommand_4

The UserDefCommand_4 property specifies the fourth parameter used in the command that is tied to a user-defined button for the CM tool. Default = empty string

UserDefCommand_4_Title

In the Configuration Items window (File > Configuration Items) there are four user defined command buttons on the lower right side. The user can specify a title for each of these four command buttons (in this case UserDefCommand_4_Title). When rolled over with the pointer, the user-defined title is displayed.

Right-clicking on a unit in the browser and selecting Configuration Management from the submenu that appears displays CM operations and user-defined titles. When a user-defined title is selected, the command value is displayed on the status bar.

UseSCCtool

The UseSCCtool property specifies whether the standard SCC interface between Rhapsody and your CM tool is used. Note that when you use the SCC interface, all of the batch mode command properties are unused.

Set this value to "Yes" if you are using SYNERGY as your CM tool. Otherwise, use the CMTool property to select the tool you are using.

Default = No

UseUnitTimeStamps

The UseUnitTimeStamps property allows the user to determine whether or not to use unit time stamps.

Default = Cleared

PVCS

The PVCS metaclass contains properties that enable you to customize your CM tool.

AddMember

The AddMember property specifies the command used to add an item to the archive.

The PVCS default value is as follows: ($SupportTreeRepository ? vcs -C"$archive" -n -T' ' -i $unit : vcs -C"$archive" -n -T' ' -i "$UnitArchiveDir"("$UnitPath"))

Archive

The Archive property specifies the archive file for a project. This property is implicitly set when you use the Connect to Archive option within Rational Rhapsody, which uses the command string specified by the ConnectToCMRepository property.

Do not set the Archive property manually.

ArchiveRoot

The ArchiveRoot property is reserved for future use.

The default is ($SupportTreeRepository ? : $ArchivePath).

ArchiveSelection

The ArchiveSelection property specifies whether the archive façade is a file or directory. Most CM tools expect a file, whereas others expect a directory.

The Browse button in the Connect to Archive window runs a different browse utility for file-based and directory-based archives. When the ArchiveSelection property is set to File, the Open dialog for files is displayed by default. In this case, you cannot select a directory as the archive.

If the archive is a directory rather than a file, set the ArchiveSelection property to Directory. The Browse for Folder window is displayed instead, which allows you to select a directory.

The possible values are as follows:

- Directory - The CM archive is a directory.
- File - The CM archive is a file. This is the default for PVCS.
- None - Neither a file nor directory is expected.

With ClearCase, the ArchiveSelection property is not set (blank). This should not be changed. ClearCase ignores the path to the archive, so the ArchiveSelection property is irrelevant. Similarly, the Browse button in the Connect to Archive dialog is disabled for ClearCase.

AskOnCheckoutReadWrite

The AskOnCheckoutReadWrite property is a Boolean value that specifies whether Rational Rhapsody should prompt users if they want to perform a check out (because they already have a read/write copy of the file).

The default value for PVCS is Yes.

AskOnLoadFromArchive

The AskOnLoadFromArchive property specifies whether to display a message when you check a file out of an archive that is not part of the current model.

If this property is set to Yes, Rational Rhapsody displays a window when you perform a checkout from the archive (or fetch operation for SCC) from either the ListArchive or Synchronize window.

When you check out a file from the archive and it is not the part of the current model, there are two possibilities:

- The corresponding parent unit is not loaded in the model.
- The parent unit, if loaded, is not the latest one.

Therefore, before Rhapsody checks out an element from an archive, it checks whether the file is already part of the model. If not, it checks whether there is a stub unit in the model that refers to the same file. If neither criteria is met, Rational Rhapsody displays an informational message.

Default = Yes

AuxProjPath

The AuxProjPath property is a string that identifies the SCC project path. Do not change this property. Removing the property value will disconnect the link between the Rational Rhapsody project and the SCC project.

If you remove this value, you must also remove the ProjName property.

Default = empty string

BaseAwareAutoMergeInvocation

The BaseAwareAutoMergeInvocation property specifies how to run the external textual DiffMerge tool which supports a base-aware detection of triviality of textual difference and a base-aware automatic merging between a base unit file and two other unit files.

An automatic model merge operation can be performed only if all the differences in the current diff session are trivial differences. If two units are being compared with a base unit, this three-way comparison makes it possible for the DiffMerge tool to determine automatically the need for some merges using the concept of trivial versus non-trivial differences. For a difference in which only one unit differs from the base unit, it is identified as a non-conflicting difference or trivial difference. Similarly, if both of the units are different from base contributor but the differences are same, then it is also a trivial difference. This applies to differences between model elements or between the attributes of model elements. However, if one unit contains a difference that does not appear in either the base unit or the other unit being compared, this is a non-trivial difference that must be resolved by the developer. During the automatic merge, all the differences is automatically accepted. There is no default value.

BaseAwareDiffInvocation

The property BaseAwareDiffInvocation specifies how to run an external textual DiffMerge tool supporting base-aware comparison and merging in Base Aware Diff mode, between a base unit file and two other unit files.

Default = Blank

BaseAwareDiffMergeInvocation

The BaseAwareDiffMergeInvocation property specifies how to run the external textual DiffMerge tool supporting a base-aware comparison and merging in Base Aware Merge mode between a base unit file and two other unit files.

CallCheckOutOnSynchronize

The CallCheckOutOnSynchronize property is a Boolean value that determines whether a checkout operation when you specify synchronization in Rational Rhapsody. Some CM tools require a full checkout of the selected elements, followed by an Add to Model operation. However, ClearCase does not require a checkout. Put a Check in this property check box to ensure a checkout operation.

Default = Cleared

CheckIn

The CheckIn property specifies the command used to check an item into the archive using the main Configuration Items window.

This command is specific to the CM tool in use.

The PVCS default is as follows:

($SupportTreeRepository ? put -C"$archive" -y $CheckInRevisionPart $mode $LogPart $unit : put -C"$archive" -y $CheckInRevisionPart $mode $LogPart "$UnitArchiveDir"("$UnitPath")).

CheckInRevisionPart

The CheckInRevision property defines a new revision number used to check in a unit using the CheckIn property. The value for $CheckInRevisionPart is taken from the Label field in the dialog for checking in a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check In the selected items.

The PVCS default value is ($label ? -v$label).

CheckOut

The CheckOut property specifies the command used to check an item out of the archive using the main Configuration Items dialog.

The PVCS default value is as follows:

($SupportTreeRepository ? get -C"$archive" -y -r$label $mode $unit : get -C"$archive" -y -r$label $mode "$UnitArchiveDir"("$UnitPath")).

CheckOutFromArchive

The CheckOutFromArchive property specifies the command used to check an item out of the archive directly using the List Archive window. For all the currently supported tools (ClearCase and PVCS Dimensions), this command is the same as that stored in the CheckOut property. Therefore, its value is $CheckOut.

Default =$CheckOut

CheckOutRevisionPart

The CheckOutRevisionPart property defines a revision number used to check out a unit using the CheckOut property. The value for $CheckOutRevisionPart is taken from the Revision/Label field in the dialog for checking out a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check Out the selected items.

Default = ($label ? -r $label : -h)

CMHeaderItsLockedBy

The CMHeaderItsLockedBy property specifies the regular expression used to extract the name of the person who last locked the unit.

For all tools, an empty string in CMHeader* means that the unit does not contain this type of information, therefore it is missing from the Configuration Items dialog when you select the List Archive option.

All properties of the format CMHeader* are regular expressions that extract header information embedded into a configuration item when it is first checked into the archive. Embedded information can include such things as the ID of the item, its revision number, and the name of the person who last locked it (the “locker”).

Header information is contained in predefined, tool-specific keywords, which the various CM tools know how to expand. For example, PVCS knows how to expand the keyword $Header: R:/StmOO/Master/cg/PropertyHelp/rcs/ConfigurationManagement.xml 1.5 2006/06/21 16:51:33 cleonardo Exp $ into a string containing the name of the working file, the version, and the locker.

The list of predefined keywords embedded in an item when it is first checked in is contained in a file stored in the HeaderFile property.

All regular expressions in CMHeader* properties match only one set of parentheses.

The PVCS default is empty string (blank).

CMHeaderItsVersion

The CMHeaderItsVersion property specifies the regular expression that extracts the version information.

Consider the following PVCS expression: \$Revision: +([0-9\.]+)

This expression searches the header for a string that begins with \$Revision: and contains a version number that can consist of one or more digits 0 through 9, the backslash character (\), or a period.

All properties of the format CMHeader* are regular expressions that extract header information embedded into a configuration item when it is first checked into the archive. Embedded information can include such things as the ID of the item, its revision number, and the name of the person who last locked it (the “locker”).

Header information is contained in predefined, tool-specific keywords, which the various CM tools know how to expand. For example, PVCS knows how to expand the keyword $Header: R:/StmOO/Master/cg/PropertyHelp/rcs/ConfigurationManagement.xml 1.5 2006/06/21 16:51:33 cleonardo Exp $ into a string containing the name of the working file, the version, and the locker.

The list of predefined keywords embedded in an item when it is first checked in is contained in a file stored in the HeaderFile property.

All regular expressions in CMHeader* properties match only one set of parentheses.

The PVCS default is \$Revision: +([0-9\.]+).

This command references the properties CheckInRevisionPart, ModePart, and LogPart and the internal variables $archive, $archivedirectory, and $unit.

CommentsRequiredForCheckIn

The CommentsRequiredForCheckIn property sets whether comments are required upon SCC Check In. When this flag is set to “True”, comments are required for successful SCC Check In operation. By default this setting is set to "False".

ConnectToCMRepository

The ConnectToCMRepository property specifies the command used to connect Rhapsody to a CM archive. For some tools, this is simply an echo.

This is a command to run the Rational Rhapsody Executer with two arguments:

- The first argument consists of two executable commands: move $rhpdirectory $rhpdirectory.orig cleartool mkelem -eltype directory -nc -nco $rhpdirectory
- The first command backs up the repository (the user’s _rpy directory). The second command is defined within ClearCase to create an element of type directory. The -nc option (no additional comment)

creates an event record with no user-supplied comment string. The new directory points to the repository.

- The second argument, "..", tells the Executer to run the commands from the directory just above the current one.

A side-effect of the ConnectToCMRepository property is that it sets the Archive property to the location of the CM archive, even if a CM command is not actually executed.

The PVCS default value is as follows:

($SupportTreeRepository ? echo "Connected to $archive" : findstr "VCSDIR" "$archive")

Delete

The Delete property specifies the script that deletes a particular item from the current ClearCase directory element.

When you delete a Rhapsody unit and the project settings indicate that the CM tool is ClearCase, Rational Rhapsody runs this delete script to remove the deleted item from the ClearCase view as well.

If the property does not exist, or if is it empty, the unit is removed from the Rational Rhapsody model, but not from the ClearCase view.

By default, this feature is disabled. To enable it, set the property DeleteActivation.

The default value of the Delete property is as follows: \"$OMROOT/etc/Executer.exe\" \"cleartool checkout -reserved -nc $dirs ; cleartool rmname -nc $units ; cleartool checkin -nc $dirs\"

It uses the following keywords:

- $units - Specifies the full path names of the unit file names, separated by spaces
- $dirs - Specifies the names of the unit directories, separated by spaces

DeleteActivation

The DeleteActivation property is a Boolean value that specifies whether deletion of units from the Rational Rhapsody model will trigger a delete command in the archive.

The possible values are as follows:

- Disable - Disable the trigger.
- UserConfirmation - Prompt the user for confirmation before performing the deletion.
- Automatic - Automatically trigger the delete command in the archive when a unit is deleted in Rational Rhapsody.

Default = Disable

DeleteDirectory

The DeleteDirectory property specifies the command to delete a directory in the Clearcase configuration management system.

The PVCS default value is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$UnitDirPath\.." ; cleartool rmname -nc "$UnitDirPath" ; cleartool checkin -nc "$UnitDirPath\.."

DiffInvocation

The DiffInvocation property specifies the command to run the external, textual DiffMerge tool.

For PVCS, this property runs the corresponding CM tool's diff tool. For the other CM tools, this property is set to an empty string.

Rhapsody searches for the property value as follows:

- First, it searches through the CM property values.
- If the property is not found or if it is set to empty, Rational Rhapsody searches under General::DiffMerge.

By default, site-specific properties override the factory properties.

A new batch file, pvcsmerge.bat, has been added to the Rational Rhapsody installation, which runs the PVCS DiffMerge tool if the CM tool is set to PVCS.

SCC users who use PVCS Dimensions™ can set this property to the PVCS value because both PVCS Dimensions and PVCS Version Manager™ use the same DiffMerge tool on Windows systems.

The PVCS default is "$OMROOT\etc\pvcsdiffmerge.bat" $source1 $source2.

DiffMergeInvocation

The DiffMergeInvocation property specifies the command to run the external, textual Diff/Merge tool.

For PVCS and Clearcase, this property runs the corresponding CM tool's diff tool. For the other CM tools, this property is set to an empty string.

Rhapsody searches for the property value as follows:

- First, it searches through the CM property values.
- If the property is not found or if it is set to empty, Rational Rhapsody searches under General::DiffMerge.

By default, site-specific properties override the factory properties.

A new batch file, pvcsmerge.bat, has been added to the Rational Rhapsody installation, which runs the PVCS Diff/Merge tool if the CM tool is set to PVCS.

SCC users who use PVCS Dimensions™ can set this property to the PVCS value because both PVCS

Dimensions and PVCS Version Manager™ use the same Diff/Merge tool on Windows systems.

The PVCS default is $DiffInvocation $output.

EnableSCCCancel

The boolean property EnableSCCCancel is used to provide a cancel option during CM operations.

When set to True, the SCC provider displays its cancel window during CM operations.

The DeleteActivation property specifies whether deleting units from the Rational Rhapsody model will trigger the delete command in the archive.

The possible values are as follows:

- Disable - The delete operation is disabled.
- UserConfirmation - The delete operation is performed only on user confirmation.
- Automatic - If you delete a unit from Rhapsody, the unit is automatically removed from the archive without any notification.

Default = Disable

Fetch

The Fetch property specifies the command used by the Rational Rhapsody Diff/Merge to fetch files from the CM tool.

The PVCS default is get -P -C"$archive" -y -r$label $mode $unit >$targetDir\$unit.

FetchFromArchive

The FetchFromArchive property specifies the command used by the Rational Rhapsody Diff/Merge to fetch files from the CM archive.

Default = $Fetch

FooterFile

The FooterFile property specifies the file footer.

The PVCS default is $OMROOT/cm/PVCSFooter.txt.

HeaderFile

The HeaderFile property specifies the file header. This property is reserved for future use.

The PVCS default is $OMROOT/cm/PVCSHeader.txt

HeaderInfoItsRepositoryPath

The HeaderInfoItsRepositoryPath property is used for internal purposes only. Do not change this value.

The default value for PVCS is as follows: VCSDIR([="0-9a-zA-Z:_/.\]+)

InValidCharactersInRevisionDescription

The InValidCharactersInRevisionDescription property provides a list of iInvalid characters in the Revision Description (or comments) during a Check In operation. The list of invalid characters - provided by the user - validates the revision description and notifies the user if there are any invalid characters.

Default = Blank

ListArchive

The ListArchive property specifies the command to list the contents of the archive.

The following example shows sample output from the ListArchive command:

version animPingPong.cfg@@\main\2 Rule: element * \main\LATEST version animPingPongFlat.cfg@@\main\2 Rule: element * \main\LATEST version animPingPongMultiThread.cfg@@\main\2 Rule: element * \main\LATEST version Default.sbs@@\main\2 Rule: element * \main\LATEST version DefaultConfig.cfg@@\main\2 Rule: element * \main\LATEST version Model1.omd@@\main\4 Rule: element * \main\LATEST version MSC1.msc@@\main\3 Rule: element * \main\LATEST

An empty string in any of the ListArchive* properties means that the configuration item does not contain this information; therefore, it is missing from the archive listing.

The PVCS default is vlog -C"$archive" $ListArchiveRevisionPart *.??v.

ListArchiveItsLockedBy

The ListArchiveItsLockedBy property specifies the regular expression that extracts from the output of the List Archive command the name of the person who last locked a configuration item.

Consider the following PVCS expression: Locked by: +([0-9a-zA-Z]+)

This expression tells the interpreter to look in the header information for a “Locked by:” string beginning. Once the string is matched, only the regular expression between the brackets is taken as the “locker.”

ListArchiveItsVersion

The ListArchiveItsVersion property specifies the version of the CM archive.

The PVCS default is Rev ([0-9\.]+).

ListArchiveItsWorkingFile

The ListArchiveItsWorkingFile specifies the regular expression that extracts from the output of the List Archive command the working file of an item.

The PVCS default is v\(([0-9a-zA-Z:_/\.\\-]+)\).

ListArchiveRevisionPart

The ListArchiveRevisionPart property is a string that represents the revision part from the output of the List Archive command.

The PVCS default is ($label ? -br $label : -br).

LockItem

The LockItem property is a string that specifies the command tool used to lock an item in the archive.

The PVCS default is as follows:

($SupportTreeRepository ? vcs -C"$archive" -y -l$label $unit : vcs -C"$archive" -y -l$label "$UnitArchiveDir"("$UnitPath")).

LogPart

The LogPart property specifies how to embed a log, if provided, in a CM command. The log is the comment entered in the Revision/Description field in the Check In window.

The PVCS default is ($log ? -m$log : -m' ').

MakeCMShadowDir

The MakeCMShadowDir property specifies the ClearCase command to create a directory as a VOB element.

The SCC default value is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$parentdir" ; move "$fulldir" "$fulldir.orig" ; cleartool mkelem -eltype directory -nc -nco "$fulldir" ; cleartool checkin -nc "$parentdir" ; copy "$fulldir.orig" "$fulldir" ; " .."

MergeOutput

The MergeOutput property specifies the file that will hold the results of a merge operation.

The PCVS default value is $temp\out.txt.

MultiRecordDelimiter

The MultiRecordDelimiter property specifies the regular expression representing the symbol used to separate multiple records in an archive listing.

The PCVS default value is =+.

OperationErrorPattern

The OperationErrorPattern property notifies you that the specified error occurred during batch mode. Rhapsody searches all CM operation output for the string specified in this property. Rhapsody first checks for errors, then for warnings.

Note the following:

- The string pattern is not case-sensitive.
- It can be a regular expression.

Default = empty string

OperationWarningPattern

The OperationWarningPattern property notifies you that the specified warning occurred during batch mode. Rhapsody searches all CM operation output for the string specified in this property. Rhapsody first checks for errors, then for warnings.

Note the following:

- The string pattern is not case-sensitive.
- It can be a regular expression.

Default = empty string

PostConnectToCMRepository

The PostConnectToCMRepository property is used for internal purposes only.The default is ($ArchiveRoot ? "$OMROOT/etc/Executer.exe" "md "$ArchiveRoot\$projectname_rpy" " :).

Do NOT change the value of this property.

ReadOnly

The ReadOnly property specifies how to embed a ReadOnly flag in the CM command.

The default value for PVCS is an empty string (blank).

ReadWrite

The ReadWrite property specifies how to embed a ReadWrite flag in the CM command.

Default = "-l"

This command references the LogPart property, which in turn references the internal variable $log, and the variable $unit.

Repository

The Repository property is used for internal purposes only. Do not change the value of this property.

The PVCS default is ($ArchiveRoot ? "$ArchiveRoot$SubDirs" :).

SaveOnCheckOut

The SaveOnCheckOut property is a Boolean value that specifies whether a Rhapsody save should be triggered whenever a checkout occurs.

Default = Checked

ShowNewItemsInSynchronize

The ShowNewItemsInSynchronize property is a Boolean value that is directly related to what you see in the Synchronize window.

If this property is set to No, new items that are added (by another member of the team) to the archive after the Rational Rhapsody project is open are not displayed.

Default = Yes

SupportTreeRepository

The SupportTreeRepository property is provided for backward compatibility to previous versions of Rational Rhapsody.

For PVCS, this property is for internal purposes only. Do not change the value of this property.

If you checked in Rational Rhapsody projects to an SCC archive prior to Version 4.0, you must perform the following steps:

- Create a directory in the CM tool with the same name of the directory that holds the .rpy file.
- Disconnect from the existing archive.
- Change the value of this property to an empty string.
- Reconnect to the archive.

The default value for PVCS is No.

UnLockItem

The UnLockItem property is a string that specifies the command used to release a lock placed on an item in the archive.

The SCC default value is as follows:

($SupportTreeRepository ? vcs -C"$archive" -y -u$label $unit : vcs -C"$archive" -y -u$label "$UnitArchiveDir"("$UnitPath"))

SCC

The SCC metaclass contains properties that enable you to use the SCC interface with Rational Rhapsody.

AddNewUnitsToArchiveDuringCheckin

This property controls whether new subunits that may have been added to a checked out unit are automatically added to your CM archive during checkin if the parent unit is checked in with descendants. When this property is set to Checked, new subunits that were added to a checked out unit are automatically added to your CM archive during checkin if the parent unit is checked in with descendants.

Default = Cleared

AskOnCheckoutReadWrite

The AskOnCheckoutReadWrite property is a Boolean value that specifies whether Rational Rhapsody should prompt users if they want to perform a check out (because they already have a read/write copy of the file).

The default value for SCC is No.

AskOnLoadFromArchive

The AskOnLoadFromArchive property specifies whether to display a message when you check a file out of an archive that is not part of the current model.

If this property is set to Yes, Rational Rhapsody displays a window when you perform a checkout from the archive (or fetch operation for SCC) from either the ListArchive or Synchronize window.

When you check out a file from the archive and it is not the part of the current model, there are two possibilities:

- The corresponding parent unit is not loaded in the model.
- The parent unit, if loaded, is not the latest one.

Therefore, before Rhapsody checks out an element from an archive, it checks whether the file is already part of the model. If not, it checks whether there is a stub unit in the model that refers to the same file. If neither criteria is met, Rational Rhapsody displays an informational message.

Default = Yes

AuxProjPath

The AuxProjPath property is a string that identifies the SCC project path. Do not change this property. Removing the property value will disconnect the link between the Rational Rhapsody project and the SCC project.

If you remove this value, you must also remove the ProjName property.

Default = empty string

BaseAwareAutoMergeInvocation

The BaseAwareAutoMergeInvocation property specifies how to run the external textual DiffMerge tool which supports a base-aware detection of triviality of textual difference and a base-aware automatic merging between a base unit file and two other unit files.

An automatic model merge operation can be performed only if all the differences in the current diff session are trivial differences. If two units are being compared with a base unit, this three-way comparison makes it possible for the DiffMerge tool to determine automatically the need for some merges using the concept of trivial versus non-trivial differences. For a difference in which only one unit differs from the base unit, it is identified as a non-conflicting difference or trivial difference. Similarly, if both of the units are different from base contributor but the differences are same, then it is also a trivial difference. This applies to differences between model elements or between the attributes of model elements. However, if one unit contains a difference that does not appear in either the base unit or the other unit being compared, this is a non-trivial difference that must be resolved by the developer. During the automatic merge, all the differences is automatically accepted. There is no default value.

BaseAwareDiffInvocation

The property BaseAwareDiffInvocation specifies how to run an external textual DiffMerge tool supporting base-aware comparison and merging in Base Aware Diff mode, between a base unit file and

two other unit files.

Default = Blank

BaseAwareDiffMergeInvocation

The BaseAwareDiffMergeInvocation property specifies how to run an external textual DiffMerge tool supporting base-aware comparison and merging in Base Aware Merge mode between a base unit file and two other unit files.

CallCheckOutOnSynchronize

The CallCheckOutOnSynchronize property is a Boolean value that determines whether a checkout operation when you specify synchronization in Rational Rhapsody. Some CM tools require a full checkout of the selected elements, followed by an Add to Model operation. However, ClearCase does not require a checkout. Put a Check in this property check box to ensure a checkout operation.

Default = Cleared

CheckIn

The CheckIn property specifies the command used to check an item into the archive using the main Configuration Items window.

This command is specific to the CM tool in use.

The PVCS default is as follows:

($SupportTreeRepository ? put -C"$archive" -y $CheckInRevisionPart $mode $LogPart $unit : put -C"$archive" -y $CheckInRevisionPart $mode $LogPart "$UnitArchiveDir"("$UnitPath")).

CheckInRevisionPart

The CheckInRevision property defines a new revision number used to check in a unit using the CheckIn property. The value for $CheckInRevisionPart is taken from the Label field in the dialog for checking in a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check In the selected items.

The SCC default value is ($label ? -v$label).

CheckOut

The CheckOut property specifies the command used to check an item out of the archive using the main Configuration Items dialog.

The SCC default value is as follows:

($SupportTreeRepository ? get -C"$archive" -y -r$label $mode $unit : get -C"$archive" -y -r$label $mode "$UnitArchiveDir"("$UnitPath")).

CheckOutFromArchive

The CheckOutFromArchive property specifies the command used to check an item out of the archive directly using the List Archive window. For all the currently supported tools (ClearCase and PVCS Dimensions), this command is the same as that stored in the CheckOut property. Therefore, its value is $CheckOut.

Default =$CheckOut

CheckOutRevisionPart

The CheckOutRevisionPart property defines a revision number used to check out a unit using the CheckOut property. The value for $CheckOutRevisionPart is taken from the Revision/Label field in the dialog for checking out a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check Out the selected items.

Default = ($label ? -r $label : -h)

CommentsRequiredForCheckIn

The CommentsRequiredForCheckIn property sets whether comments are required upon SCC Check In. When this flag is set to Checked, comments are required for successful SCC Check In operation. The default is Cleared.

ConnectToCMRepository

The ConnectToCMRepository property specifies the command used to connect Rhapsody to a CM archive. For some tools, this is simply an echo.

This is a command to run the Rational Rhapsody Executer with two arguments:

- The first argument consists of two executable commands: move $rhpdirectory $rhpdirectory.orig cleartool mkelem -eltype directory -nc -nco $rhpdirectory
- The first command backs up the repository (the user’s _rpy directory). The second command is defined within ClearCase to create an element of type directory. The -nc option (no additional comment) creates an event record with no user-supplied comment string. The new directory points to the repository.
- The second argument, "..", tells the Executer to run the commands from the directory just above the current one.

A side-effect of the ConnectToCMRepository property is that it sets the Archive property to the location of the CM archive, even if a CM command is not actually executed.

The SCC default value is as follows:

($SupportTreeRepository ? echo "Connected to $archive" : findstr "VCSDIR" "$archive")

Delete

The Delete property specifies the script that deletes a particular item from the current ClearCase directory element.

When you delete a Rhapsody unit and the project settings indicate that the CM tool is ClearCase, Rational Rhapsody runs this delete script to remove the deleted item from the ClearCase view as well.

If the property does not exist, or if is it empty, the unit is removed from the Rational Rhapsody model, but not from the ClearCase view.

By default, this feature is disabled. To enable it, set the property DeleteActivation.

The default value of the Delete property is as follows: \"$OMROOT/etc/Executer.exe\" \"cleartool checkout -reserved -nc $dirs ; cleartool rmname -nc $units ; cleartool checkin -nc $dirs\"

It uses the following keywords:

- $units - Specifies the full path names of the unit file names, separated by spaces
- $dirs - Specifies the names of the unit directories, separated by spaces

DeleteActivation

The DeleteActivation property is a Boolean value that specifies whether deletion of units from the Rational Rhapsody model will trigger a delete command in the archive.

The possible values are as follows:

- Disable - Disable the trigger.
- UserConfirmation - Prompt the user for confirmation before performing the deletion.
- Automatic - Automatically trigger the delete command in the archive when a unit is deleted in Rational Rhapsody.

Default = Disable

DeleteDirectory

The DeleteDirectory property specifies the command to delete a directory in the Clearcase configuration management system.

The SCC default value is as follows:

"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$UnitDirPath\.." ; cleartool rmname -nc "$UnitDirPath" ; cleartool checkin -nc "$UnitDirPath\.."

DiffInvocation

The DiffInvocation property specifies the command to run the external, textual DiffMerge tool.

For SCC, this property is set to an empty string.

Rhapsody searches for the property value as follows:

- First, it searches through the CM property values.
- If the property is not found or if it is set to empty, Rational Rhapsody searches under General::DiffMerge.

By default, site-specific properties override the factory properties.

A new batch file, pvcsmerge.bat, has been added to the Rational Rhapsody installation, which runs the PVCS DiffMerge tool if the CM tool is set to PVCS.

SCC users who use PVCS Dimensions™ can set this property to the PVCS value because both PVCS Dimensions and PVCS Version Manager™ use the same DiffMerge tool on Windows systems.

Default = Blank

DiffMergeInvocation

The DiffMergeInvocation property specifies the command to run an external, textual Diff/Merge tool.

SCC users who use PVCS Dimensions™ can set this property to the PVCS value because both PVCS Dimensions and PVCS Version Manager™ use the same Diff/Merge tool on Windows systems.

EnableSCCCancel

The boolean property EnableSCCCancel is used to provide a cancel option during CM operations.

When set to Checked, the SCC provider displays its cancel window during CM operations.

The DeleteActivation property specifies whether deleting units from the Rational Rhapsody model will trigger the delete command in the archive.

The possible values are as follows:

- Cleared - The delete operation is disabled.
- Checked - If you delete a unit from Rhapsody, the unit is automatically removed from the archive without any notification.

Default = Cleared

IgnoreAddToArchiveForExistingUnits

This property controls whether the Add to Archive operation in SCC mode is ignored if units already exist in your CM archive. When this property is set to Checked, the Add to Archive command is ignored if units already exist in your CM archive. A message displays on the Configuration Management tab of the Output window to this effect.

If you set the property to Cleared, a pop-up error message box may open instead, which you will have to close before you can continue.

Default = Checked

IgnoreUndoCheckoutForNotCheckedoutUnits

This property controls whether the Undo Checkout operation in SCC mode is ignored if a unit is not already checked out. When this property is set to Checked, the Undo Checkout operation is ignored if a unit is not already checked out. A message displays on the Configuration Management tab of the Output window to this effect.

If you set the property to Cleared, a pop-up error message box may open instead, which you will have to close before you can continue.

Default = Checked

MergeOutput

The MergeOutput property specifies the file that will hold the results of a merge operation.

The SCC default value is empty string.

MoveActivation

The MoveActivation property is a Boolean value that specifies whether moving units in the Rational Rhapsody model (in a way that changes the unit file location on the hard drive) will trigger a rename command in the archive.

The possible values are as follows:

- Disable - Disable this functionality.
- UserConfirmation - Prompt the user for confirmation before renaming the elements in the archive.
- Automatic - Automatically rename elements whenever units are moved in the Rational Rhapsody model.

Default = Disable

ProjName

The ProjName property is a string that identifies the SCC project name. Do not change this property. Removing the property value will disconnect the link between the Rational Rhapsody project and the SCC

project.

If you remove this value, you must also remove the AuxProjPath property.

Default = empty string

RedirectOutputToRhapsody

The UseSCCtool property is a Boolean value that specifies whether the resulting output of an SCC CM command should be redirected (displayed) in the Rational Rhapsody CM output window tab.

Default = Checked

RefreshCMStatusAtProjectOpenup

RefreshCMStatusAtProjectOpenup is a property that determines whether or not the CM status of project units is updated from the CM repository when a project is opened. This property is set at the project level. The value of this property is only relevant when the property ConfigurationManagement::SCC::ShowCMStatus is set to true.

The possible values are Yes, No, and Ask User.

Default = Ask User

RenameActivation

The RenameActivation property is a Boolean value that specifies whether renaming units in the Rational Rhapsody model will trigger a delete command in the archive.

The possible values are as follows:

- Disable - Disable this functionality.
- UserConfirmation - Prompt the user for confirmation before deleting the elements from the archive.
- Automatic - Automatically delete the element units that are renamed in the Rational Rhapsody model.

Default = Disable

SaveOnCheckOut

The SaveOnCheckOut property is a Boolean value that specifies whether a Rhapsody save should be triggered whenever a checkout occurs.

Default = Checked

ShowCMStatus

The property ShowCMStatus is a boolean property that determines whether or not Rational Rhapsody displays the CM status of project units. This property is set at the project level.

Default = Cleared

StoreInSeparateDirectoryActivation

The StoreInSeparateDirectoryActivation property affects how an existing package is converted. The following conditions apply with this property only if the SCC tool is ClearCase:

- Enable this property by selecting UserConfirmation or Automatic from the drop-down list. When an existing flat package is converted to a package as a directory, the directory is created on the configuration management side and the children of this package are moved to this directory.
- Disable this property by selecting Disable. When an existing flat package is converted to a package as a directory, the directory is removed on the configuration management side and the children of this package are removed as well.

Default = Disable

SupportTreeRepository

The SupportTreeRepository property is provided for backward compatibility to previous versions of Rational Rhapsody.

For PVCS, this property is for internal purposes only. Do not change the value of this property.

If you checked in Rational Rhapsody projects to an SCC archive prior to Version 4.0, you must perform the following steps:

- Create a directory in the CM tool with the same name of the directory that holds the .rpy file.
- Disconnect from the existing archive.
- Change the value of this property to an empty string.
- Reconnect to the archive.

The default for SCC is an empty string.

UnrollLoops

When you perform a CM operation, this property specifies whether Rational Rhapsody initiates a single SCC call or multiple SCC calls for the operation.

If the property is enabled, a single SCC call is issued.

SourceIntegrity

Contains properties that control the interaction of Rational Rhapsody with the SourceIntegrity configuration management tool.

AddMember

The AddMember property specifies the command used to add an item to the archive.

The default is ($SupportTreeRepository ? "$OMROOT/etc/Executer.exe" "copy $unit \"$archivedirectory\\\.\"; pj add -y -P \"$archive\" \"$archivedirectory\$unit\" : "$OMROOT/etc/Executer.exe" "copy $UnitPath \"$archivedirectory$UnitDirectory\\\.\"; pj add -y -P \"$archive\" \"$archivedirectory$UnitDirectory\$unit\")

Archive

The Archive property specifies the archive file for a project. This property is implicitly set when you use the Connect to Archive option within Rational Rhapsody, which uses the command string specified by the ConnectToCMRepository property.

Do not set the Archive property manually.

Default = $currentdirectory

ArchiveSelection

The ArchiveSelection property specifies whether the archive façade is a file or directory. Most CM tools expect a file, whereas others expect a directory.

The Browse button in the Connect to Archive window runs a different browse utility for file-based and directory-based archives. When the ArchiveSelection property is set to File, the Open dialog for files is displayed by default. In this case, you cannot select a directory as the archive.

If the archive is a directory rather than a file, set the ArchiveSelection property to Directory. The Browse for Folder window is displayed instead, which allows you to select a directory.

The possible values are as follows:

- Directory - The CM archive is a directory.
- File - The CM archive is a file. This is the default for SourceIntegrity.
- None - Neither a file nor directory is expected.

AskOnCheckoutReadWrite

The AskOnCheckoutReadWrite property is a Boolean value that specifies whether Rational Rhapsody should prompt users if they want to perform a check out (because they already have a read/write copy of the file).

Default = Yes

AskOnLoadFromArchive

The AskOnLoadFromArchive property specifies whether to display a message when you check a file out of an archive that is not part of the current model.

If this property is set to Yes, Rational Rhapsody displays a window when you perform a checkout from the archive (or fetch operation for SCC) from either the ListArchive or Synchronize window.

When you check out a file from the archive and it is not the part of the current model, there are two possibilities:

- The corresponding parent unit is not loaded in the model.
- The parent unit, if loaded, is not the latest one.

Therefore, before Rhapsody checks out an element from an archive, it checks whether the file is already part of the model. If not, it checks whether there is a stub unit in the model that refers to the same file. If neither criteria is met, Rational Rhapsody displays an informational message.

Default = Yes

BaseAwareAutoMergeInvocation

The BaseAwareAutoMergeInvocation property specifies how to run the external textual DiffMerge tool which supports a base-aware detection of triviality of textual difference and a base-aware automatic merging between a base unit file and two other unit files.

An automatic model merge operation can be performed only if all the differences in the current diff session are trivial differences. If two units are being compared with a base unit, this three-way comparison makes it possible for the DiffMerge tool to determine automatically the need for some merges using the concept of trivial versus non-trivial differences. For a difference in which only one unit differs from the base unit, it is identified as a non-conflicting difference or trivial difference. Similarly, if both of the units are different from base contributor but the differences are same, then it is also a trivial difference. This applies to differences between model elements or between the attributes of model elements. However, if one unit contains a difference that does not appear in either the base unit or the other unit being compared, this is a non-trivial difference that must be resolved by the developer. During the automatic merge, all the differences is automatically accepted. There is no default value.

BaseAwareDiffInvocation

The property BaseAwareDiffInvocation specifies how to run an external textual DiffMerge tool supporting base-aware comparison and merging in Base Aware Diff mode, between a base unit file and two other unit files.

Default = Blank

BaseAwareDiffMergeInvocation

The BaseAwareDiffMergeInvocation property specifies how to run an external textual DiffMerge tool

supporting base-aware comparison and merging in Base Aware Merge mode between a base unit file and two other unit files.

CMHeaderItsLockedBy

The CMHeaderItsLockedBy property specifies the regular expression used to extract the name of the person who last locked the unit.

For all tools, an empty string in CMHeader* means that the unit does not contain this type of information, therefore it is missing from the Configuration Items dialog when you select the List Archive option.

All properties of the format CMHeader* are regular expressions that extract header information embedded into a configuration item when it is first checked into the archive. Embedded information can include such things as the ID of the item, its revision number, and the name of the person who last locked it (the "locker").

Header information is contained in predefined, tool-specific keywords, which the various CM tools know how to expand. For example, PVCS knows how to expand the keyword $Header: R:/StmOO/Master/cg/PropertyHelp/rcs/ConfigurationManagement.xml 1.5 2006/06/21 16:51:33 cleonardo Exp $ into a string containing the name of the working file, the version, and the locker.

The list of predefined keywords embedded in an item when it is first checked in is contained in a file stored in the HeaderFile property.

All regular expressions in CMHeader* properties match only one set of parentheses.

CMHeaderItsVersion

The CMHeaderItsVersion property specifies the regular expression that extracts the version information.

Consider the following PVCS expression: \$Revision: +([0-9\.]+)

This expression searches the header for a string that begins with \$Revision: and contains a version number that can consist of one or more digits 0 through 9, the backslash character (\), or a period.

All properties of the format CMHeader* are regular expressions that extract header information embedded into a configuration item when it is first checked into the archive. Embedded information can include such things as the ID of the item, its revision number, and the name of the person who last locked it (the "locker").

Header information is contained in predefined, tool-specific keywords, which the various CM tools know how to expand. For example, PVCS knows how to expand the keyword $Header: R:/StmOO/Master/cg/PropertyHelp/rcs/ConfigurationManagement.xml 1.5 2006/06/21 16:51:33 cleonardo Exp $ into a string containing the name of the working file, the version, and the locker.

The list of predefined keywords embedded in an item when it is first checked in is contained in a file stored in the HeaderFile property.

All regular expressions in CMHeader* properties match only one set of parentheses.

This command references the properties CheckInRevisionPart, ModePart, and LogPart and the internal variables $archive, $archivedirectory, and $unit.

CheckIn

The CheckIn property specifies the command used to check an item into the archive using the main Configuration Items window.

This command is specific to the CM tool in use.

The default is as follows:

($SupportTreeRepository ? pj ci -y -t " $CheckInRevisionPart $ModePart $LogPart -P "$archive" -w . "$archivedirectory/$unit" : pj ci -y -t " $CheckInRevisionPart $ModePart $LogPart -P "$archive" -w . "$archivedirectory$UnitDirectory/$unit").

CheckInRevisionPart

The CheckInRevision property defines a new revision number used to check in a unit using the CheckIn property. The value for $CheckInRevisionPart is taken from the Label field in the dialog for checking in a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check In the selected items.

The default is ($label ? -N $label).

CheckOut

The CheckOut property specifies the command used to check an item out of the archive using the main Configuration Items dialog.

The default value is as follows:

($SupportTreeRepository ? pj co -y $CheckOutRevisionPart $ModePart -P "$archive" -w . "$archivedirectory/$unit" : pj co -y $CheckOutRevisionPart $ModePart -P "$archive" -w . "$archivedirectory$UnitDirectory/$unit").

CheckOutFromArchive

The CheckOutFromArchive property specifies the command used to check an item out of the archive directly using the List Archive window. For all the currently supported tools (ClearCase and PVCS Dimensions), this command is the same as that stored in the CheckOut property. Therefore, its value is $CheckOut.

Default =$CheckOut

CheckOutRevisionPart

The CheckOutRevisionPart property defines a revision number used to check out a unit using the CheckOut property. The value for $CheckOutRevisionPart is taken from the Revision/Label field in the dialog for checking out a unit that appears after selecting FileConfiguration Items, selecting a unit, and then selecting Check Out the selected items.

Default = ($label ? -r $label : -h)

ConnectToCMRepository

The ConnectToCMRepository property specifies the command used to connect Rhapsody to a CM archive. For some tools, this is simply an echo.

This is a command to run the Rational Rhapsody Executer with two arguments:

- The first argument consists of two executable commands:
- move $rhpdirectory $rhpdirectory.orig
- cleartool mkelem -eltype directory -nc -nco $rhpdirectory

The first command backs up the repository (the user's _rpy directory). The second command is defined within ClearCase to create an element of type directory. The -nc option (no additional comment) creates an event record with no user-supplied comment string. The new directory points to the repository.

The second argument, "..", tells the Executer to run the commands from the directory just above the current one.

A side-effect of the ConnectToCMRepository property is that it sets the Archive property to the location of the CM archive, even if a CM command is not actually executed.

DiffInvocation

The DiffInvocation property specifies the command to run the external, textual DiffMerge tool.

For SourceIntegrity, this property is set to an empty string.

Rhapsody searches for the property value as follows:

- First, it searches through the CM property values.
- If the property is not found or if it is set to empty, Rational Rhapsody searches under General::DiffMerge.

By default, site-specific properties override the factory properties.

Default = Blank

DiffMergeInvocation

The DiffMergeInvocation property specifies the command to run an external, textual Diff/Merge tool.

SCC users who use PVCS Dimensions™ can set this property to the PVCS value because both PVCS Dimensions and PVCS Version Manager™ use the same Diff/Merge tool on Windows systems.

Fetch

The Fetch property specifies the command used by the Rational Rhapsody Diff/Merge to fetch files from the CM tool.

The default is pj co -y -p $CheckOutRevisionPart $ModePart -P "$archive" -w . "$archivedirectory$UnitDirectory/$unit" >$targetDir\$unit.

FetchFromArchive

The FetchFromArchive property specifies the command used by the Rational Rhapsody Diff/Merge to fetch files from the CM archive.

Default = $Fetch

FooterFile

The FooterFile property specifies the file footer.

The default is $OMROOT/cm/SIFooter.txt.

HeaderFile

The HeaderFile property specifies the file header. This property is reserved for future use.

The default is $OMROOT/cm/SIHeader.txt.

HeaderInfoItsRepositoryPath

The HeaderInfoItsRepositoryPath property is used for internal purposes only. Do not change this value.

The default value is as follows: Repository: ([0-9a-zA-Z:_/.\]+)

InValidCharactersInRevisionDescription

The InValidCharactersInRevisionDescription property provides a list of iInvalid characters in the Revision Description (or comments) during a Check In operation. The list of invalid characters - provided by the user - validates the revision description and notifies the user if there are any invalid characters. By default this property value is blank.

ListArchive

The ListArchive property specifies the command to list the contents of the archive.

The following example shows sample output from the ListArchive command:

version animPingPong.cfg@@\main\2 Rule: element * \main\LATEST version animPingPongFlat.cfg@@\main\2 Rule: element * \main\LATEST version animPingPongMultiThread.cfg@@\main\2 Rule: element * \main\LATEST version Default.sbs@@\main\2 Rule: element * \main\LATEST version DefaultConfig.cfg@@\main\2 Rule: element * \main\LATEST version Model1.omd@@\main\4 Rule: element * \main\LATEST version MSC1.msc@@\main\3 Rule: element * \main\LATEST

An empty string in any of the ListArchive* properties means that the configuration item does not contain this information; therefore, it is missing from the archive listing.

ListArchiveItsLockedBy

The ListArchiveItsLockedBy property specifies the regular expression that extracts from the output of the List Archive command the name of the person who last locked a configuration item.

Consider the following expression: Locked by: +([0-9a-zA-Z]+)

This expression tells the interpreter to look in the header information for a “Locked by:” string beginning. Once the string is matched, only the regular expression between the brackets is taken as the “locker.”

ListArchiveItsVersion

The ListArchiveItsVersion property specifies the version of the CM archive.

The default is Revision: ([0-9\.]+).

ListArchiveItsWorkingFile

The ListArchiveItsWorkingFile specifies the regular expression that extracts from the output of the List Archive command the working file of an item.

The default is Archive File: ([0-9a-zA-Z: _/\.\\-]+).

ListArchiveRevisionPart

The ListArchiveRevisionPart property is a string that represents the revision part from the output of the List Archive command.

The default is ($label ? -r $label : -h).

LockItem

The LockItem property is a string that specifies the command tool used to lock an item in the archive.

The default is pj lock -y $CheckOutRevisionPart -P "$archive" "$archivedirectory/$unit".

LogPart

The LogPart property specifies how to embed a log, if provided, in a CM command. The log is the comment entered in the Revision/Description field in the Check In window.

The default is ($log ? -m$log : -m ").

MergeOutput

The MergeOutput property specifies the file that will hold the results of a merge operation.

The default value is an empty string (blank).

ModePart

The ModePart property specifies the locking mode of a configuration item. This is defined as: $mode

If the item is locked, $mode is replaced by the value of the ReadWrite property; otherwise, it is replaced by the value of the ReadOnly property.

Move

The Move property specifies the ClearCase command for a unit move.

The default value is as follows: "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$olddir" "$newdir"; cleartool mv -nc "$oldName" "$newName" ; cleartool checkin -nc "$olddir" "$newdir" "

MultiRecordDelimiter

The MultiRecordDelimiter property specifies the regular expression representing the symbol used to separate multiple records in an archive listing.

Default = The

OperationErrorPattern

The OperationErrorPattern property notifies you that the specified error occurred during batch mode. Rhapsody searches all CM operation output for the string specified in this property. Rhapsody first checks for errors, then for warnings.

Note the following:

- The string pattern is not case-sensitive.
- It can be a regular expression.

Default = empty string

OperationWarningPattern

The OperationWarningPattern property notifies you that the specified warning occurred during batch mode. Rhapsody searches all CM operation output for the string specified in this property. Rhapsody first checks for errors, then for warnings.

Note the following:

- The string pattern is not case-sensitive.
- It can be a regular expression.

Default = empty string

ReadOnly

The ReadOnly property specifies how to embed a ReadOnly flag in the CM command.

The default value is an empty string (blank).

ReadWrite

The ReadWrite property specifies how to embed a ReadWrite flag in the CM command.

Default = "-l"

This command references the LogPart property, which in turn references the internal variable $log, and the variable $unit.

Repository

The Repository property is used for internal purposes only. Do not change the value of this property.

SaveOnCheckOut

The SaveOnCheckOut property is a Boolean value that specifies whether a Rhapsody save should be

triggered whenever a checkout occurs.

Default = Checked

SupportTreeRepository

The SupportTreeRepository property is provided for backward compatibility to previous versions of Rational Rhapsody.

If you checked in Rational Rhapsody projects to an SCC archive prior to Version 4.0, you must perform the following steps:

- Create a directory in the CM tool with the same name of the directory that holds the .rpy file.
- Disconnect from the existing archive.
- Change the value of this property to an empty string.
- Reconnect to the archive.

Default = No

UnLockItem

The UnLockItem property is a string that specifies the command used to release a lock placed on an item in the archive.

The default value is as follows:

(pj unlock -y $CheckOutRevisionPart -P "$archive" "$archivedirectory/$unit"

Synergy

These properties control the interaction of Rational Rhapsody with the SYNERGY configuration management system.

AssignedTasksItsTaskId

This property specifies the regular expression that extracts the Task ID from the output of the ListAssignedTasks command.

When you create a task, Rational Synergy names it, by default, as Task task_number. However, when you configure your DCM server, you can set it to insert a prefix before task_number. In this case, you may want to update this property to use the regular expression you want. For example, if the prefix used is ukan#, then the value of this property should be changed to Task ukan#([0-9\.]+); otherwise the default is Task ([0-9\.]+).

The default is Task ([0-9\.]+).

AssignedTasksItsTitle

This property specifies the regular expression that extracts the Task Title from the output of the ListAssignedTasks command.

The default is Task (.*).

CheckinCurrentTask

This property specifies the command used to check in the current (default) SYNERGY Task.

Default = "$OMROOT/etc/Executer.exe" "ccm task -checkin default -comment \"default task checked in from Rhapsody\" -y"

CreateTask

This property specifies the command used to create a SYNERGY Task.

Default = "$OMROOT/etc/Executer.exe" "ccm task -create -gui"

GetCurrentTask

This property specifies the command used to get current (default) SYNERGY Task of the user.

Default = ccm task -default

GetCurrentTaskItsTaskId

This property specifies the regular expression that extracts the Task ID from the output of the GetCurrentTask command.

The default is (([^#]*#)?[0-9.]+).

ListAssignedTasks

This property specifies the command used to list the SYNERGY Tasks assigned to the current user.

Default = "$OMROOT/etc/Executer.exe" "ccm task -query -task_scope all_my_assigned"

LoadTaskOnOpenProject

If this value is set to Checked, Rational Rhapsody loads the task list when opening a project. If the property is set to Cleared, use the refresh button to load the tasks after loading the project.

This is useful when loading the tasks slows down “open project” because of the CM server is in a remote location and loading the task takes a while. The default value of the property is Checked.

MultiRecordDelimiter

This property specifies the regular expression representing the symbol used to separate multiple records in an archive listing.

Default = Empty string

SetCurrentTask

This property specifies the command used to set current (default) SYNERGY Task.

Default = "$OMROOT/etc/Executer.exe" "ccm task -default $TaskId"

ViewTask

This property specifies the command used to view a SYNERGY Task.

Default = "$OMROOT/etc/Executer.exe" "ccm task -gui $TaskId"

CORBA

The CORBA subject enables you to use CORBA™ attributes with Rational Rhapsody Developer for C++. The metaclasses are as follows:

- C++Mapping_CORBABasic
- C++Mapping_CORBAEnum
- C++Mapping_CORBAFixedArray
- C++Mapping_CORBAFixedSequence
- C++Mapping_CORBAFixedStruct
- C++Mapping_CORBAFixedUnion
- C++Mapping_CORBAInterfaceReference
- C++Mapping_CORBAInterfaceVariable
- C++Mapping_CORBASequence
- C++Mapping_CORBAVariableArray
- C++Mapping_CORBAVariableStruct
- C++Mapping_CORBAVariableUnion
- Class
- Configuration
- Operation
- Package
- TAO
- Type
- UserDefinedORB

The CORBA subject is available only in Rational Rhapsody Developer for C++.

C++Mapping_CORBABasic

The C++Mapping_CORBABasic metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = $MappedType)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)

C++Mapping_CORBAEnum

The C++Mapping_CORBAEnum metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = $MappedType)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)

C++Mapping_CORBAFixedArray

The C++Mapping_CORBAFixedArray metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType slice)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType slice)*

C++Mapping_CORBAFixedSequence

The C++Mapping_CORBAFixedSequence metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType&)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)*

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)*

C++Mapping_CORBAFixedStruct

The C++Mapping_CORBAFixedStruct metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType&)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)

C++Mapping_CORBAFixedUnion

The C++Mapping_CORBAFixedUnion metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType&)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)

C++Mapping_CORBAInterfaceReference

The C++Mapping_CORBAInterfaceReference metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = $MappedType)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)

C++Mapping_CORBAInterfaceVariable

The C++Mapping_CORBAInterfaceVariable metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = $MappedType)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)

C++Mapping_CORBASequence

The C++Mapping_CORBASequence metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType&)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)*

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)*

C++Mapping_CORBAVariableArray

The C++Mapping_CORBAVariableArray metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType slice&)*

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType slice)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType slice)*

C++Mapping_CORBAVariableStruct

The C++Mapping_CORBAVariableStruct metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType&)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)*

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)*

C++Mapping_CORBAVariableUnion

The C++Mapping_CORBAVariableUnion metaclass contains properties that affect how CORBA stereotypes are mapped to C++ code.

in

The "in" property specifies how the CORBA type "in" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG®.

(Default = const $MappedType&)

inout

The "inout" property specifies how the CORBA type "inout" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)

out

The "out" property specifies how the CORBA type "out" is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType&)*

ReturnValue

The ReturnValue property specifies how a return value that is a CORBA type is mapped to C++ code. The default mapping is defined by the IDL to C++ Language Mapping!EF specification from the OMG.

(Default = $MappedType)*

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

(Default = $MappedType)*

Class

The Class metaclass contains properties that affect CORBA classes.

C++Implementation

The property C++Implementation allows the user to select the CORBA reference interface or the CORBA variable interface (_ptr and _var classes), for mapping during code generation.

The type of interface selected determines the metaclass that is used.

The possible values are Reference (default) and Variable.

DefaultImplementationMethod

The DefaultImplementationMethod property specifies which method is used to implement CORBA interfaces.

The possible values are as follows:

- Inheritance - Use inheritance as the implementation method.
- TIE - Use TIE as the implementation method.

(Default = Inheritance)

IDLSequence

The property IDLSequence determines the name of the typedef used in the implementation of to-many relations.

(Default = $interfaceSeq)

InheritanceRealizes

The InheritanceRealizes property enables you to override the default implementation method for CORBA interfaces, as specified by the DefaultImplementationMethod property, for a particular class.

To implement a CORBA interface using inheritance, if the default implementation method is set to TIE , set the InheritanceRealizes property for the realizing class to the names of the CORBA interfaces that it should realize.

(Default = empty string)

InstanceNameInConstructor

The InstanceNameInConstructor property specifies whether to add a string parameter representing the instance name to all class constructors.

(Default = Cleared)

TIERealizes

The TIERealizes property enables you to override the default implementation method for CORBA interfaces, as specified by the DefaultImplementationMethod property, for a particular class.

To implement a CORBA interface using TIE, if the default implementation method is set to Inheritance , set the TIERealizes property for the realizing class to the names of the CORBA interfaces that it should realize.

(Default = empty string)

Configuration

The Configuration metaclass contains properties that control the CORBA configuration.

CORBAEnable

The CORBAEnable property specifies whether to generate code for a CORBA client, CORBA server, or neither.

The possible values are as follows:

- No - Generate code for neither a client nor a server.
- CORBAClient - Generate code for a CORBA client.
- CORBAServer - Generate code for a CORBA server.

(Default = No)

ExposeCorbaInterfaces

The ExposeCorbaInterfaces property generates server IDL code for the specified CORBA interfaces.

(Default = empty string)

IDLExtension

The IDLExtension property specifies the extension for IDL files.

(Default = .idl)

IncludeIDL

The IncludeIDL property is a string that lists the IDL files to include at the component level. Separate multiple files with commas.

(Default = empty string)

ORB

The ORB property specifies the ORB with which you are working. The value is either TAO or UserDefinedORB - Use this setting to add a new ORB.

(Default = TAO)

StartFrameworkInMainThread

The StartFrameworkInMainThread property is a Boolean value that specifies whether the framework should control the main thread. When you set this property to True at the configuration level, OXF::start(FALSE) is called and the OXF takes over the main thread.

You would use this property in the case where you want Product; to take over the main thread instead of CORBA.

(Default = Cleared)

UseCorbaInterfaces

The UseCorbaInterfaces property generates client IDL code for the specified CORBA interfaces.

(Default = empty string)

Operation

The Operation metaclass contains properties that affect CORBA operations.

C++DefaultThrow

(Default = CORBA::SystemException)

IsOneWay

The IsOneWay property specifies whether to create a OneWay CORBA operation.

(Default = Cleared)

ThrowExceptions

The ThrowExceptions property specifies the exceptions that an operation can throw. Separate multiple exceptions with commas.

(Default = empty string)

Package

The Package metaclass contains a property that controls interface packages.

DeclareInterfacesInModule

The DeclareInterfacesInModule property specifies whether forward declaration of CORBA interfaces is enabled.

(Default = Cleared)

TAO

The TAO metaclass contains properties that affect CORBA TAO.

AddCORBAEnvParam

The AddCORBAEnvParam property specifies whether to add a CORBA environment parameter (of type CORBA_env) as the last argument to CORBA operations.

(Default = Cleared)

ClientMainLineTemplate

The ClientMainLineTemplate property enables you to add code in the main function of a CORBA client.

(Default = empty MultiLine)

CORBAIncludePath

The CORBAIncludePath property specifies the location of CORBA include files.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

The default value is as follows: $(ACE_ROOT)\TAO\ $(ACE_ROOT)\ $(ACE_ROOT)\TAO\orbsvcs

CORBALibs

The CORBALibs property specifies the locations of the CORBA libraries.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

The default value is as follows: $(ACE_ROOT)\TAO\tao\PortableServer\TAO_PortableServerd.lib $(ACE_ROOT)\TAO\tao\Valuetype\TAO_Valuetyped.lib $(ACE_ROOT)\TAO\tao\TAOd.lib $(ACE_ROOT)\ace\aced.lib

CPP_CompileSwitches

The CPP_CompileSwitches property is a string that enables you to specify additional compiler switches.

The default value is as follows: /GR /D "ACE_AS_STATIC_LIBS" /D "TAO_AS_STATIC_LIBS"

CPP_LinkSwitches

The CPP_LinkSwitches property is a string that enables you to specify additional link switches, needed when your component is linked with this ORB’s libraries.

(Default = empty string)

CPP_StandardInclude

The CPP_StandardInclude property is a string that enables you to specify additional header files to be included in the generated sources, needed when your component is compiled with this ORB’s include files.

(Default = tao/CORBA.h;tao/PortableServer/POA.h)

DefTIEString

The DefTIEString property specifies a template for the string generated into every IDL file that contains a CORBA interface, if the DefaultImplementationMethod property is set to TIE .

The default value is blank.

DestroyInitialInstance

The DestroyInitialInstance property specifies a template that destroys the object created during the initial instance.

(Default = orb-destroy();)

EnvParamDefaultVal

The EnvParamDefaultVal property is a string that specifies an environment parameter as the last argument to operations that implement CORBA interfaces.

The default value is CORBA::default_environment.

EnvParamName

The EnvParamName property is a string that specifies the name of the environment parameter added by the EnvParamDefaultVal property.

(Default = IT_env)

EnvParamType

The EnvParamType property specifies the type of the environment parameter added by the EnvParamDefaultVal property.

(Default = CORBA::Environment&)

IDLCompileCommand

The IDLCompileCommand property specifies the compile command for a given IDL compiler.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

The default is as follows:

@echo IDL Compiling $OMFileSpecPath $(ACE_ROOT)\bin\tao_idl -o $OMFileSpecDir

IDLCompileSwitches

The IDLCompileSwitches property specifies the switches for the IDL compiler.

There are two ways to glue a CORBA implementation class to the ORB - BOA and TIE. The Rhapsody default is BOA. The TAO -B flag compiles the IDL so BOA objects are created.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

(Default = -B)

ImplementationExtension

The ImplementationExtension property specifies the extension for implementation files.

(Default = .cpp)

InitialInstance

The InitialInstance property specifies any additional initial instance routines required by the ORB. This code template is generated for each instance of a specific class (implementing one or more CORBA interfaces). Note: the template is inserted when the class was selected in the Explicit Initial Instances in the Configuration dialog, but not for user-created instances.

For example: /****** Default TAO InitialInstance **********/ try {

// If you open this commented code, the createRefFile should be defined in $instance

// PortableServer::ServantBase_var servant = $instance;

// $instance-createRefFile(orb);

}

catch(const CORBA::Exception e) {

omcerr "Got CORBA exception in instanciation" omendl;

omcerr e omendl;

return 1;

}

Note that $ClassName will expand to the name of the class being initialized and $instance will expand to its instance name.

(Default = above example)

InitializeORB

The InitializeORB property specifies the ORB initialization routines. In most cases, this is the first executable command in the main function of the CORBA server.

You can place any ORB initialization code in this property. For example: "/* Default Initializing ORB and getting POA Manager */ // any ORB initialization PortableServer::POAManager_var poa_manager = NULL; CORBA::ORB_var orb = NULL; PortableServer::POA_var rootPOA = NULL; CORBA::PolicyList policies = NULL; try { // Initialize the ORB. orb = CORBA::ORB_init(argc, argv); // get a reference to the root POA CORBA::Object_var obj = orb-resolve_initial_references(\"RootPOA\"); rootPOA = PortableServer::POA::_narrow(obj); policies.length(1); policies[(CORBA::ULong)0] = rootPOA-create_lifespan_policy(PortableServer::PERSISTENT); // get the POA Manager poa_manager = rootPOA-the_POAManager(); } catch(const CORBA::Exception e) { cerr \"Got CORBA exception in initialization\" endl; cerr e endl; return 1; }

"

*(Default = /****** Default TAO Initilizing ORB and getting POA Manager **********/*

CORBA::ORB_var orb = NULL;

CORBA::Object_var poaObj = NULL;

PortableServer::POA_var rootPoa = NULL;

PortableServer::POAManager_var manager = NULL;

try {

// Initialize the ORB.

orb = CORBA::ORB_init(argc, argv);

poaObj = orb - resolve_initial_references("RootPOA");

rootPoa = PortableServer::POA::_narrow(poaObj.in());

manager = rootPoa - the_POAManager();

}

catch(const CORBA::Exception e) {

omcerr "Got CORBA exception in initialization" omendl;

omcerr e omendl;

return 1;

}

)

The default value for VisiBrokerRT is “ .cc ”; the default value for all other ORBs is “.cpp” .

NeededObjForClient

The NeededObjForClient property is an enumerated type that specifies the file needed to create an object.

(Default = Stub)

NeededObjForClientServer

The NeededObjForClientServer property is an enumerated type that specifies the file needed to create a client server. The possible values are as follows:

- Stub
- Skeleton
- Both

(Default = Both)

NeededObjForServer

The NeededObjForServer property is an enumerated type that specifies the file needed to create a server. The possible values are as follows:

- Stub
- Skeleton
- Both

(Default = Both)

ServerMainLineTemplate

The ServerMainLineTemplate property is a MultiLine type that defines how Tao interacts with Rational Rhapsody. This property is used in the “main” part of the generated application.

The default value for TAO is as follows:

```
/****** Default TAO Server Mainline **********/

try {

// Activate POA Manager

manager-activate();

// Wait for incoming requests

orb-run();

}

catch(const CORBA::Exception e) {

omcerr "Got CORBA exception in orb run" omendl;

omcerr e omendl;

return 1;
```

}

Skeleton

The Skeleton property specifies the inheritance format used by a given ORB vendor.

(Default = POA_$interface)

The CORBA interface name replaces the $interface variable in the generated code.

SkeletonImplementationName

The SkeletonImplementationName property is a string that defines the naming behavior for skeleton implementation files.

(Default = $interfaceS)

SkeletonSpecificationName

The SkeletonSpecificationName property is a string that defines the naming behavior for skeleton specification files.

(Default = $interfaceS)

SpecificationExtension

The SpecificationExtension property is a string that specifies the extension for specification files.

(Default = .h)

StubImplementationName

The StubImplementationName property is a string that defines the naming behavior for stub implementation files.

(Default = $interfaceC)

StubSpecificationName

The StubSpecificationName property is a string that defines the naming behavior for stub specification files.

(Default = $interfaceC)

Type

The Type metaclass contains properties that enable you to change the OMG default mappings.

C++Implementation

The property C++Implementation allows the user to select the CORBA fixed construct or the CORBA variable construct, for mapping during code generation.

The type of construct selected determines the metaclass that is used.

The possible values are Fixed and Variable (default).

CORBAStereotype

The CORBAStereotype property specifies the CORBA stereotype that is applied to a CORBA type.

The possible values are as follows:

CORBABasic

CORBAEnum

CORBAFixedArray

CORBAFixedSequence

CORBAFixedStruct

CORBAFixedUnion

CORBAInterfaceReference

CORBAInterfaceVariable

CORBASequence

CORBAVariableArray

CORBAVariableStruct

CORBAVariableUnion

(Default = CORBABasic)

CPP_in

The CPP_in property is a string that overrides the OMG default IDL to C++ language mapping for a CORBA in parameter.

(Default = empty string)

CPP_inout

The CPP_inout property is a string that overrides the OMG default IDL to C++ language mapping for a CORBA inout parameter.

(Default = empty string)

CPP_out

The CPP_out property is a string that overrides the OMG default IDL to C++ language mapping for a CORBA out parameter.

(Default = empty string)

CPP_return_value

The CPP_return_value property is a string that overrides the OMG default IDL to C++ language mapping for a CORBA return value type.

(Default = empty string)

IDLSequence

The property IDLSequence determines the name of the typedef used in the implementation of to-many relations.

(Default = $typeSeq)

UserDefinedORB

The UserDefinedORB metaclass contains properties that affect CORBA TAO.

AddCORBAEnvParam

The AddCORBAEnvParam property specifies whether to add a CORBA environment parameter (of type CORBA_env) as the last argument to CORBA operations.

(Default = Checked)

ClientMainLineTemplate

The ClientMainLineTemplate property enables you to add code in the main function of a CORBA client.

(Default = empty MultiLine)

CORBAIncludePath

The CORBAIncludePath property specifies the location of CORBA include files.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

The default value is as follows: $(IT_CONFIG_PATH)\..\include

CORBALibs

The CORBALibs property specifies the locations of the CORBA libraries.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

The default value is as follows: $(IT_CONFIG_PATH)\..\lib\ITMi.lib

CPP_CompileSwitches

The CPP_CompileSwitches property is a string that enables you to specify additional compiler switches.

CPP_LinkSwitches

The CPP_LinkSwitches property is a string that enables you to specify additional link switches, needed when your component is linked with this ORB’s libraries.

(Default = empty string)

CPP_StandardInclude

The CPP_StandardInclude property is a string that enables you to specify additional header files to be included in the generated sources, needed when your component is compiled with this ORB’s include

files.

(Default = CORBA.h)

DefTIEString

The DefTIEString property specifies a template for the string generated into every IDL file that contains a CORBA interface, if the DefaultImplementationMethod property is set to TIE .

The default value is as follows: DEF_TIE_$interface($class)

DestroyInitialInstance

The DestroyInitialInstance property specifies a template that destroys the object created during the initial instance.

The default value is orb-destroy();

EnvParamDefaultVal

The EnvParamDefaultVal property is a string that specifies an environment parameter as the last argument to operations that implement CORBA interfaces.

The default value is CORBA::default_environment .

EnvParamName

The EnvParamName property is a string that specifies the name of the environment parameter added by the EnvParamDefaultVal property.

(Default = IT_env)

EnvParamType

The EnvParamType property specifies the type of the environment parameter added by the EnvParamDefaultVal property.

(Default = CORBA::Environment &)

IDLCompileCommand

The IDLCompileCommand property specifies the compile command for a given IDL compiler.

This property is visible both in the Properties window and on the Middleware tab of the Configuration

window in the browser.

(Default = idl)

IDLCompileSwitches

The IDLCompileSwitches property specifies the switches for the IDL compiler.

There are two ways to glue a CORBA implementation class to the ORB - BOA and TIE. The Rhapsody default is BOA. The TAO -B flag compiles the IDL so BOA objects are created.

This property is visible both in the Properties window and on the Middleware tab of the Configuration window in the browser.

(Default = -B)

ImplementationExtension

The ImplementationExtension property specifies the extension for implementation files.

(Default = .cpp)

InitialInstance

The InitialInstance property specifies any additional initial instance routines required by the ORB. This code template is generated for each instance of a specific class (implementing one or more CORBA interfaces).

For example: /****** Default TAO InitialInstance **********/

try {

// If you open this commented code, the createRefFile should be defined in $instance

// PortableServer::ServantBase_var servant = $instance;

// $instance-createRefFile(orb);

}

catch(const CORBA::Exception e) {

omcerr "Got CORBA exception in instanciation" omendl;

omcerr e omendl;

return 1;

}

Note that $ClassName will expand to the name of the class being initialized and $instance will expand to its instance name.

(Default = above example)

InitializeORB

The InitializeORB property specifies the ORB initialization routines. In most cases, this is the first executable command in the main function of the CORBA server.

You can place any ORB initialization code in this property. For example: "/* Default Initializing ORB and getting POA Manager */ // any ORB initialization PortableServer::POAManager_var poa_manager = NULL; CORBA::ORB_var orb = NULL; PortableServer::POA_var rootPOA = NULL; CORBA::PolicyList policies = NULL; try { // Initialize the ORB. orb = CORBA::ORB_init(argc, argv); // get a reference to the root POA CORBA::Object_var obj = orb-resolve_initial_references(\"RootPOA\"); rootPOA = PortableServer::POA::_narrow(obj); policies.length(1); policies[(CORBA::ULong)0] = rootPOA-create_lifespan_policy(PortableServer::PERSISTENT); // get the POA Manager poa_manager = rootPOA-the_POAManager(); } catch(const CORBA::Exception e) { cerr \"Got CORBA exception in initialization\" endl; cerr e endl; return 1; }

"

*(Default = /****** Default TAO Initilizing ORB and getting POA Manager **********/*

CORBA::ORB_var orb = NULL;

CORBA::Object_var poaObj = NULL;

PortableServer::POA_var rootPoa = NULL;

PortableServer::POAManager_var manager = NULL;

try {

// Initialize the ORB.

orb = CORBA::ORB_init(argc, argv);

poaObj = orb - resolve_initial_references("RootPOA");

rootPoa = PortableServer::POA::_narrow(poaObj.in());

manager = rootPoa - the_POAManager();

}

catch(const CORBA::Exception e) {

```
omcerr "Got CORBA exception in initialization" omendl;

omcerr e omendl;

return 1;

}

)
```

The default value for VisiBrokerRT is “ .cc ”; the default value for all other ORBs is “.cpp” .

NeededObjForClient

The NeededObjForClient property is an enumerated type that specifies the file needed to create an object.

(Default = Stub)

NeededObjForClientServer

The NeededObjForClientServer property is an enumerated type that specifies the file needed to create a client server. The possible values are as follows:

- Stub
- Skeleton
- Both

(Default = Skeleton)

NeededObjForServer

The NeededObjForServer property is an enumerated type that specifies the file needed to create a server. The possible values are as follows:

- Stub
- Skeleton
- Both

(Default = Skeleton)

ServerMainLineTemplate

The ServerMainLineTemplate property is a MultiLine type that defines how Tao interacts with Rational Rhapsody. This property is used in the “main” part of the generated application.

The default value for TAO is as follows:

```
/****** Default TAO Server Mainline **********/

try {

// Activate POA Manager

manager-activate();

// Wait for incoming requests

orb-run();

}

catch(const CORBA::Exception e) {

omcerr "Got CORBA exception in orb run" omendl;

omcerr e omendl;

return 1;

}
```

Skeleton

The Skeleton property specifies the inheritance format used by a given ORB vendor.

(Default = $interfaceBOAImpl)

The CORBA interface name replaces the "$interface" variable in the generated code.

SkeletonImplementationName

The SkeletonImplementationName property is a string that defines the naming behavior for skeleton implementation files.

(Default = $interfaceS)

SkeletonSpecificationName

The SkeletonSpecificationName property is a string that defines the naming behavior for skeleton specification files.

(Default = $interface)

SpecificationExtension

The SpecificationExtension property is a string that specifies the extension for specification files.

(Default = .hh)

StubImplementationName

The StubImplementationName property is a string that defines the naming behavior for stub implementation files.

(Default = $interfaceC)

StubSpecificationName

The StubSpecificationName property is a string that defines the naming behavior for stub specification files.

(Default = $interface)

CPP_CG

The CPP_CG subject contains several metaclasses for operating system environments and the following general metaclasses:

- Argument
- Attribute
- Borland
- Class
- Configuration
- Cygwin
- Dependency
- Event
- File
- Framework
- General
- Generalization
- INTEGRITY
- INTEGRITY5
- Integrity5ESTL
- IntegrityESTL
- Linux
- Microsoft
- MicrosoftDLL
- MicrosoftWinCE600
- ModelElement
- MSStandardLibrary
- Multi4Win32
- MultiWin32
- NucleusPLUS-PCC
- Operation
- OsePPCDiab
- OseSfk
- Package
- Port
- QNXNeutrinoMomentics
- QNXNeutrinoGCC
- Relation

- Solaris2
- Solaris2GNU
- Statechart
- Type
- VxWorks
- VxWorks6diab
- VxWorks6diab_RTP
- VxWorks6gnu
- VxWorks6gnu_RTP
- WorkbenchManaged
- WorkbenchManaged_RTP

Argument

The Argument metaclass contains properties that control how arguments are generated in code.

ClassWide

The ClassWide property determines whether a class-wide modifier is generated for the argument.

Default = False

DeclarationModifier

The property DeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in argument declarations. Keywords that appear between the argument type and the argument name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and PostDeclarationModifier.

Default = Blank

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules.

The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified element’s tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

IsRegister

The property IsRegister can be used to specify that the keyword "register" should be generated in the code for a given argument.

Default = Cleared

IsVolatile

The property IsVolatile allows you to specify that a specific operation argument should be declared as volatile.

Default = Cleared

PostDeclarationModifier

The property PostDeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in argument declarations. Keywords that appear after the argument name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and DeclarationModifier.

Default = Blank

PreDeclarationModifier

The property PreDeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in argument declarations. Keywords that appear before the argument type are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties DeclarationModifier and PostDeclarationModifier.

Default = Blank

PrintName

When an operation argument is not accessed in the operation body, C++ compilers will issue a warning. The property PrintName allows you to avoid such warnings by having the Rational Rhapsody -generated code include only the argument type but not the argument name for such arguments.

If set to True, the argument name is included in the generated code. If set to False, only the argument type is included in the generated code.

Default = Checked.

Attribute

The Attribute metaclass contains properties that control attributes of code generation, such as whether to generate accessor operations.

Accessor

The Accessor property is ignored by Rhapsody.

AccessorGenerate

The AccessorGenerate property specifies whether to generate accessor operations for attributes. The possible values are as follows:

- Checked - A get() method is generated for the attribute. This is the default value for C++.
- Cleared - A get() method is not generated for the attribute. This is the default value for C.

Setting this property to Cleared is one way to optimize your code for size.

AccessorVisibility

The AccessorVisibility property specifies the access level of the generated accessor for attributes. This enables you to define the access level of an accessor operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute access level for the accessor.
- public - Set the accessor access level to public.
- private - Set the accessor access level to private.
- protected - Set the accessor access level to protected.

Default = fromAttribute

AttributeInitializationFile

The AttributeInitializationFile property specifies how static const attributes are initialized. In Rhapsody, you can initialize these attributes in the specification file or directly in the initialization file. This property is analogous to the VariableInitializationFile property for global const variables.

The possible values are as follows:

- Default - The attribute is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize constant attributes in the implementation file.
- Specification - Initialize constant attributes in the specification file.

Default = Default

BitField

Allows you to define a bit field for an attribute. To define a bit field, open the Features dialog for the relevant attribute and enter the number you want to use for the bit field as the value of the property BitField.

For example, if you enter 2 as the value of BitField for an attribute named attribute_1 of type int, the resulting code is:

int attribute_1 : 2;

ConstantVariableAsDefine

This property is a Boolean value that determines whether the variable, defined as constant in file or package, is generated using a #define macro. Otherwise, it is generated using the const qualifier.

Default = Cleared

DeclarationModifier

The property DeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in attribute declarations. Keywords that appear between the attribute type and the attribute name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and PostDeclarationModifier.

Default = Blank

DeclarationPosition

The DeclarationPosition property enables you to control the declaration order of attributes. The possible values are as follows:

- Default - Similar to the AfterClassRecord setting, with the following difference:
- For static attributes defined in a class with the property CPP_CG::Attribute::Visibility set to Public, these attributes are generated after types whose CPP_CG::Type::Visibility property is set to Public.
- You should not use this setting for new models. See the Rational Rhapsody Developer for Ada documentation for more information.
- BeforeClassRecord - Generate the attribute immediately before the class record.
- AfterClassRecord - Generate the attribute immediately after the class record.
- StartOfDeclaration - Generate the attribute immediately after the start of the section (private or public part of the specification, or package body).
- EndOfDeclaration - Generate the attribute immediately before the end of the section (private or public part of the specification, or package body).

Default = Default

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An

empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument's description Operation Primitive operations, triggered operations, $Arguments - The operation argument's description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified element's tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

EnableInitializationStyleForStaticAttributes

The style used for initializing C++ attributes is determined by the property CPP_CG::Attribute::InitializationStyle, which can take the values ByInitializer (default value) and ByAssignment. The property EnableInitializationStyleForStaticAttributes allows you to specify whether or not Rational Rhapsody should apply this property to static attributes as well.

If the value of this property is set to True, the code for initializing static attributes is based on the value of the property InitializationStyle.

If the value of this property is set to False, then the value of the property InitializationStyle will have no effect on the initialization of static attributes. Rather, the initialization of static attributes will always be by assignment.

Default = True

GenerateVariableHelpers

By default, Rational Rhapsody generates getter and setter methods for class attributes, but not for global variables. If you want Rhapsody to generate getter and setter methods for global variables, set the value of the property GenerateVariableHelpers to True.

Default = Cleared

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationName

The ImplementationName property enables you to give an operation one model name and generate it with another name. It is introduced as a workaround that enables you to generate const and non-const operations with the same name. For example:

- Create a class A.
- Add a non-const operation f().
- Add a const operation f_const().
- Set the CPP_CG::Operation::ImplementationName property for f_const() to “f.”
- Generate the code.

The resulting code is as follows: class A { ... void f(); /* the non const f */ ... void f() const; /* actually f_const() */ ... }; The creation of two operations with the same signature, differing only in whether it is a const, is a common practice in C++, especially for STL users.

Default = Empty string

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef_DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

InitializationStyle

The InitializationStyle property specifies the initialization style used for attributes. When you specify an initial value for an attribute, Rational Rhapsody initializes the attribute based on the value of this property. In Rational Rhapsody Developer for C++, the possible values are as follows:

- ByInitializer - Initialize the attribute in the initializer (a(y)). This is the default value. If the initialization style is ByInitializer, the attribute initialization should be done after the user initializer, in the same order as the order of attributes in the code.
- ByAssignment - Initialize the attribute in the constructor body (a = y).

Default = ByInitializer

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for C++ The possible values for the Inline property are as follows:

- none - The operation is not generated inline.
- in_header - The operation is generated inline in the specification file.
- in_source - The operation is generated inline in the implementation file.
- in_declaration - A class operation is generated inline in the class declaration. A global function is

generated inline in the package specification file.

Inlining an operation in the header might cause problems if the function body refers to other classes. For example, if the inlined code refers to another class (via a pointer such as itsRelatedClass), inlined code generated in a header might not compile.

The implementation file for the class would have an #include for RelatedClass, but the specification file would not.

The workaround is to create a Usage dependency of the class with the inlined function on the related class. This forces an #include of the related class to be generated in the header of the dependent class with the inlined function.

Default = none

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased.

Default = False

IsMutable

The boolean property IsMutable allows you to specify that an attribute is a mutable attribute.

Default = Cleared

IsVolatile

The property IsVolatile allows you to specify that an attribute should be declared as volatile.

Default = Cleared

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java).

In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip.

When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually.

The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = None

MutatorGenerate

The MutatorGenerate property specifies whether to generate mutators for attributes. The possible values are as follows:

- Smart - Mutators are not generated for attributes that have the Constant modifier.
- Always - Mutators are generated, regardless of the modifier.
- Never - Mutators are not generated.

Default = Smart

MutatorVisibility

The MutatorVisibility property specifies the access level of the generated mutator for attributes. This enables you to define the access level of a mutator operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute's access level for the mutator.
- public - Set the mutator access level to public.
- private - Set the mutator access level to private.
- protected - Set the mutator access level to protected. This value is not available in Rational Rhapsody Developer for C.
- default - Set the mutator access level to default. This value is available only in Rational Rhapsody Developer for Java.

Default = fromAttribute

PostDeclarationModifier

The property PostDeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in attribute declarations. Keywords that appear after the attribute name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and DeclarationModifier.

Default = Blank

PreDeclarationModifier

The property PreDeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in attribute declarations. Keywords that appear before the attribute type are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties DeclarationModifier and PostDeclarationModifier.

Default = Blank

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code. See the Rational Rhapsody Help for detailed information about using composite types.

*Default = **

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

VariableInitializationFile

The VariableInitializationFile property specifies how global constant variables are initialized. You can initialize these variables in the specification file. You can use these variables as compile-time constants that can be used to define array sizes, for example. Rhapsody automatically identifies constant variables

with const. By modifying this property, you can choose the initialization file directly. The possible values are as follows:

- Default - The variable is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize global constant variables in the implementation file.
- Specification - Initialize global constant variables in the specification file.

Default = Default

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The Visibility setting has the following applicability:

- Classes - Applies only to nested classes, which are defined inside other classes.
- Types - Applies only to types that are defined inside classes. It does not apply to global types, which are defined in packages.

The following table lists the visibility for the CPP_CG subject.

- protected - Attribute is visible only within the scope of its class and descendants.
- private - Attribute is visible only within its class.
- public - Attribute is visible everywhere.
- fromAttribute - Attribute visibility depends on the Access selection in the Browser dialog, which specifies the visibility of accessors and mutators for an attribute.

Default = protected

Class

The Class metaclass contains properties that affect the generated classes.

AccessTypeName

The AccessTypeName property specifies the name of the access type generated for the class record.

Default = Empty string

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active classes. The possible values are as follows:

- A string - Specifies the message queue size for an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

Default = Empty string

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects. The possible values are as follows:

- Any integer - Specifies that a stack of that size is allocated for active objects.
- An empty string (blank) - If not specified, the stack size is set in an operating system-specific manner, based on the value of the ActiveStackSize property for the framework.

Default = Empty string

ActiveThreadName

The ActiveThreadName property indicates the real OS task or thread name. This property only matters when the class is set to active. This facilitates debugging in complex environments in which many threads are constantly being created and deleted on-the-fly. This property is effective for all targets. All strings entered must be enclosed in quotes (" "). The possible values are as follows:

- A string - Names the active thread.
- An empty string (blank) - The value is set in an operating system-specific manner, based on the value of the ActiveThreadName property for the framework.

Default = Empty string (OS selects thread name)

ActiveThreadPriority

The ActiveThreadPriority property specifies the priority of active class threads. The possible values are as follows:

- A string - Specifies thread priority of an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

Default = Empty string

AdditionalBaseClasses

The AdditionalBaseClasses property enables you to add inheritance from external classes to the model.

Default = Empty string

AdditionalNumberOfInstances

The AdditionalNumberOfInstances property is a string that specifies the size of the local heap allocated for events when the current pool is full. Triggered operations use the event properties.

This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during runtime. All events are dynamically allocated during initialization.

Once allocated, an event queue for a thread remains static in size. The possible values are as follows:

- An empty string (blank) - No additional memory is to be allocated when the initial memory pool is exhausted.
- n (a positive integer) - Specifies the size of the array allocated for additional instances.

Default = Empty string

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation.

The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to Cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to Cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to Cleared, all the arguments are not animated.
- If the AnimateArguments property is set to Cleared, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = Checked

BaseNumberOfInstances

The BaseNumberOfInstances property is a string that specifies the size of the local heap memory pool allocated for either:

- Instances of the class (CPP_CG::Class)
- Instances of the event (CPP_CG::Event)
- This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during run time. All instances of events are dynamically allocated during initialization. Once allocated, a thread’s event queue remains static in size.

Triggered operations use the properties defined for events. When the memory pool is exhausted, an additional amount, specified by the AdditionalNumberOfInstances property, is allocated. Memory pools for classes can be used only with the Flat statechart implementation scheme. The possible values are as follows:

- An empty string (blank) - Memory is always dynamically allocated.

- BaseNumberOfInstances - An array is allocated in this size for instances.

The related properties are as follows:

- AdditionalNumberOfInstances - Specifies the number of instances to allocate if the pool runs out.
- ProtectStaticMemoryPool - Specifies whether the pool should be protected (to support a multithreaded environment)
- EmptyMemoryPoolCallback - Specifies a user callback function to be called when the pool is empty. This property should be used instead of the AdditionalNumberOfInstance property for error handling.
- EmptyMemoryPoolMessage - When set to true, this property causes a message to be displayed if the pool runs out of memory in instrumented mode.

Default = Empty string

CodeGeneratorTool

The CodeGeneratorTool property specifies which code generation tool to use for the given configuration. The possible values are as follows:

- Classic - refers to the older "non-respect" code generation tool.
- Advanced - refers to the Rational Rhapsody newer code-respect-oriented code generation tool.
- External - instructs Rhapsody to use the registered external code generator.

Default = Advanced

ComplexityForInlining

The ComplexityForInlining property specifies the upper bound for the number of lines in user code that are allowed to be inlined. User code is the action part of transitions in statecharts.

For example, using the value 3, all transitions with actions consisting of three lines or fewer of code are automatically inlined in the calling function.

Inlining is replacing a function call in the generated code with the actual code statements that make up the body of the function. This optimizes the code execution at the expense of an increase in code size.

For example, increasing the number of lines that can be inlined from 3 to 5 has shortened the code execution time in some cases up to 10%. This property applies only to the Flat implementation scheme for statecharts.

Default = 0

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass

that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to “MYDLL_API.” The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class.

Default = Empty string

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified element’s tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

Destructor

The Destructor property controls the generation of virtual destructors in C++. The property exists for C for historical reasons, with a single value of auto, but it has no effect on the generated C code. The possible values are as follows:

- auto - A virtual destructor is generated for an object only if it has at least one virtual function.
- virtual - A virtual destructor is generated in all cases.

- abstract - A virtual destructor is generated as a pure virtual function.
- common - A nonvirtual destructor is generated.

Default = auto

Embeddable

The Embeddable property is a Boolean property that specifies whether a class can be allocated by value (nested) inside another class or package. For example, if the Embeddable property is True, 20 instances of a class A can be allocated inside another class using the following syntax: A itsA[20]; The possible values are as follows:

- Checked - The object can be allocated by value inside a composite object or package. The object declaration and definition are generated in the specification file of the composite.
- Cleared - The object cannot be embedded inside another object (not supported in RiC). The object declaration and definition are generated in the implementation file of the composite.

The Embeddable property is used with the EmbeddedScalar and EmbeddedFixed properties to determine how to generate code for an embedded object. It is also closely related to the ImplementWithStaticArray property, which also needs to be set in order to support by-value allocation.

Relations can be generated by value only under the following circumstances:

- The multiplicity of the relation is well-defined (not “*”).
- The ImplementWithStaticArray property of the component relation is set to FixedAndBounded.

When the Embeddable property is Cleared (RiC only):

- The attributes of the object are encapsulated. Clients of the object are forced to use it only via its operations, because there is no direct access to its attributes.
- Dynamic allocation must be used. The compiler does not know how to statically allocate an object when its declaration is not visible.

Default = Cleared

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- Checked - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for the object or object type.
- Cleared - Events are dynamically allocated during initialization, but not during runtime. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during runtime.

Default = Checked

EnableUseFromCPP

The EnableUseFromCPP property specifies whether to wrap C operations with an appropriate extern C {} wrapper to prevent problems when code is compiled with a C++ compiler. Wrapping C code with extern C enables you to include C code in a C++ application.

Note that the structure definition for the object is not wrapped - only the functions are.

For example, if the EnableUseFromCPP is set to Checked for an object, the following wrapper code is generated for its operations:

#ifdef __cplusplus extern "C" { #endif /* __cplusplus */ /* Operations */ #ifdef __cplusplus } #endif /* __cplusplus */

Default = Cleared

Final

The Final property, when set to False, specifies that the generated record for the class is a tagged record. This property applies to Ada95.

Default = False

Friend

The Friend property specifies friends to be added to class declarations. For example, if you specify “int t(); class x”, the following lines is generated in the public section of all class declarations: friend int t(); friend class x; Separate multiple friends with semicolons.

Default = Empty string

GenClassAsStruct

When generating C++ code, Rational Rhapsody generates classes in your model as C++ classes in the code. While this is the default behavior, it is also possible to have Rhapsody generate classes as structs in your C++ code. The property GenClassAsStruct allows you to specify that a class should be generated as a struct.

Default = False

GenerateAccessType

The GenerateAccessType property determines which access types are generated for the class. The possible values are as follows:

- None - Access types are not generated.
- Standard - An access type is generated.
- General - General access types are generated.

Default = General

GenerateDestructor

The GenerateDestructor property specifies whether to generate a destructor for a class.

Default = Checked

GenerateRecordType

The GenerateRecordType property determines whether the class record is generated. Default = True

HasUnknownDiscriminant

The HasUnknownDiscriminant property determines whether an unknown discriminant) is generated for this class. Default = False

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces.

Default = Empty string

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body. Default = Empty MultiLine

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body. Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

In

The In property specifies how code is generated when the type is used with an argument that has the modifier In. When a class is used with the "In" modifier, the default is "const $type&" in C++.

InitCleanUpRelations

The InitCleanUpRelations property specifies whether to generate initRelations() and cleanUpRelations() operations for sets of related global instances. This property applies only to composites and global relations. Default = True

InitializationCode

The InitializationCode property adds the specified initialization code in the body of the class. A non-abstract class can have initialization code that is executed during elaboration of the associated package. (empty MultiLine)

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier InOut. When a class is used with the "InOut" modifier, the default is "$type&" in C++.

InstanceDeclaration

The InstanceDeclaration property specifies how instances are declared in code. The default value for C is as follows: struct $cname$suffix In the generated code, the variable $cname is replaced with the object (or object type) name. The variable $suffix is replaced with the type suffix "_t," if the object is of implicit type.

Default = $cname$suffix

IsCompletedOperation

The IsCompletedOperation specifies whether state_IS_COMPLETED operations are generated as functions or macros (using #define). The possible values are as follows:

- Plain - state_IS_COMPLETED operations are generated as functions (pre-V4.2 behavior). This is the default value.
- Inline - state_IS_COMPLETED operations are generated using #define macros, if the body contains only a return statement.

Default = Plain

IsInOperation

The IsInOperation specifies how state_IN methods are generated.

In Rational Rhapsody Developer for C++, this property specifies whether state_IN methods are virtual or nonvirtual. For classes with state machines (statecharts or activity diagrams), Rational Rhapsody generates state_IN operations.

By default, these operations are nonvirtual, but you can make them virtual by setting this property to Virtual. This value is needed to support statechart inheritance in Flat mode.

Default = Default

IsLimited

The IsLimited property determines whether the class or record type is generated as limited. Default =

False

IsNested

The IsNested property specifies whether to generate the class or package as nested. Default = False

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private. Default = False

IsReactiveInterface

The IsReactiveInterface property modifies the way reactive classes are generated. It has the following effects:

- Virtual inheritance from OMReactive
- Prevents instrumentation
- Prevents the thread argument and the initialization code (setting the active context) in the class constructor
- Creates a pure-virtual destructor (by default)

This property affects only classes that declare themselves as interfaces by having a stereotype with a name that contains the word “interface” (case-insensitive).

In previous versions of Rational Rhapsody, a class could inherit from a single reactive class only, regardless of whether it was an interface or implementation class. Beginning with Version 4.0.1 MR2, a class can inherit (implement) several reactive interfaces.

In Rational Rhapsody Developer for C++, you must explicitly designate reactive interfaces because the code generator applies special translation rules involving multiple inheritance from the Rational Rhapsody framework. You can designate a reactive interface in two ways:

- Set the property CPP_CG::Class::IsReactiveInterface to checked.
- Use the predefined stereotype Reactive_interface. This stereotype uses stereotype-based code generation in order to automatically apply the correct property value.

Alternatively, you can define another stereotype (such as PortSpec) that sets IsReactiveInterface to true and use that stereotype. A class is considered reactive if it meets all the following conditions:

- The CPP_CG::Framework::ReactiveBase property is not empty.
- The CPP_CG::Framework::ReactiveBaseUsage property is set to Checked.
- One or more of the following conditions are checked:
- The class has a statechart or activity diagram.
- The class is a composite class.
- The class has event receptions or triggered operations.

Default = Cleared

Rational Rhapsody Developer for C++ A reactive interface:

- Automatically uses virtual inheritance from its base reactive class.
- Does not override reactive methods (such as startBehavior().)
- Does not call reactive initialization methods (such as setThread()).
- By default, has a pure virtual destructor (when the CPP_CG::Class::Destructor property is set to auto).
- Cannot have a statechart or activity diagram.
- Cannot be a composite class.

A class that inherits from a reactive interface:

- Overrides reactive methods (such as startBehavior().)
- Ignores the reactive interfaces when overriding reactive methods. It calls reactive initialization methods directly (such as calling setThread() in its constructor).

MangleNestedInAnimation

The boolean property MangleNestedInAnimation is used to specify that when nested classes are animated, the name used for the inner class should be mangled. For example, if class B is nested in class A, the name A_B would be used.

(Default = Cleared)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between

the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. Default = None

MaximumPendingEvents

The MaximumPendingEvents property specifies the maximum number of events that can be simultaneously pending in the event queue of the active class. The possible values are as follows:

- –1 - Memory is dynamically allocated.
- Positive integer - Specifies the maximum number of events.

Default = –1

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package. Default = Public

ObjectTypeAsSingleton

The ObjectTypeAsSingleton property enables you to generate singleton code for object-types and actors. This functionality enables you to save a singleton-type (actor) in its own repository unit, and manage that unit using a configuration management tool. Set this property for a single object-type or higher. An object-type is generated as a singleton when all of the following conditions are met:

- The object-type has the «Singleton» stereotype.
- There is one and only one object of the object-type and the object multiplicity is 1.
- The ObjectTypeAsSingleton property is set to True.

Note that when you expose a singleton object (for example, by creating a singleton object-type), Rational Rhapsody also modifies the code generated for the singleton. Default = False

OptimizeStatechartsWithoutEventsMemoryAllocation

The OptimizeStatechartsWithoutEventsMemoryAllocation property determines whether the generated code uses dynamic memory allocation for statecharts that use only triggered operations. Default = False

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out."

When a class is used with the "Out" modifier, the default is "$type*&" in C++.

ReactiveInterfaceScheme

The property ReactiveInterfaceScheme determines which framework class serves as the base class for a reactive interface.

If the property value is set to Full, the interface inherits from OMReactive.

If the property value is set to Thin, the interface inherits from IOxfEventSender.

Note that IOxfEventSender includes only operations related to event sending, while OMReactive includes also attributes and operations related to statechart behavior.

(Default = Full)

ReactiveThreadSettingPolicy

The ReactiveThreadSettingPolicy property enables you to specify how threads are set for reactive classes. The possible values are as follows:

- Default - During code generation, Rational Rhapsody adds a thread argument to the constructor.
- MainThread - Rational Rhapsody does not add an argument; the thread is set to the main thread.
- UserDefined - Rational Rhapsody does not add an argument; you must set the value for the thread yourself.

Default = Default

RecordTypeName

The RecordTypeName property specifies the name of the class record type. If this is not set, Rational Rhapsody uses class_name_t. Default = Empty string

RelativeEventDataRecordTypeComponentsNaming

The RelativeEventDataRecordTypeComponentsNaming property enables relative naming of event data record type components that represent events and triggered operation parameters. If this is True, no events or triggered operations will share argument names because they would generate record components with the same name (which would not compile). Default = False

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The

signatures of the two operations must match.

Default = Empty string

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type.

When a class is used with the "ReturnType" modifier, the default is "$type*" in C++.

SingletonExposeThis

The SingletonExposeThis property, when set to False, specifies that all non-static methods are considered as static methods and will not have a this parameter passed in. Default = False

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification. Default = Empty MultiLine

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification. Default = Empty MultiLine

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when

the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces. Default = Empty string

TaskBody

The TaskBody property enables you to define an alternate task body for Ada Task and Ada Task Type classes. Default = Empty string

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return."

*Default = $type**

See also:

- In
- InOut
- Out

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

See "Visibility" for more information.

Default = Public

Configuration

The Configuration metaclass contains properties that affect the configuration.

ClassStateDeclaration

The ClassStateDeclaration property supports C compilers that cannot handle enum declarations inside struct declaration. The possible values are as follows:

- InClassDeclaration - Generate the reactive statechart enum declaration in the class declaration (as in Rational Rhapsody 3.0.1).
- BeforeClassDeclaration - Generate the reactive class statechart enum declaration before the declaration of the class.

Default = InClassDeclaration

CodeGenerationDirectoryLevel

The property CodeGenerationDirectoryLevel is found in the pre-72 compatibility profiles for C and C++.

Before version 7.2 of Rational Rhapsody, the directories specified with the properties DefaultSpecificationDirectory and DefaultImplementationDirectory were created at the beginning of the path to the generated files, for example, ..\spec_directory\package_a\subpackage_1 and ..\impl_directory\package_a\subpackage_1.

Beginning with version 7.2 of Rational Rhapsody, the directories specified with DefaultSpecificationDirectory and DefaultImplementationDirectory are created at the end of the path to the generated files, for example, ..\package_a\subpackage_1\spec_directory and ..\package_a\subpackage_1\impl_directory.

To provide the old code generation behavior for pre-72 models, the compatibility profiles include the property CodeGenerationDirectoryLevel, with the default value of the property set to Top. If you want your pre-72 models to use the new behavior that was introduced in version 7.2, change the value of this property to Bottom.

Default = Top

CodeGeneratorTool

The property CodeGeneratorTool specifies which code generation tool to use for the given configuration. The possible values are as follows:

- Advanced - Rational Rhapsody uses its internal code generator is used to generate code

- External - instructs Rhapsody to use the registered external code generator

Default = Advanced

ContainerSet

The ContainerSet property specifies the container set used to implement relations. The possible C++ values are as follows:

OMContainers (default) OMCorba2CorbaContainers OMCpp2CorbaContainers OMCppOfCorbaContainers OMUContainers STLContainers

DefaultActiveGeneration

The DefaultActiveGeneration property specifies whether the default active class is created, as well as the classes for which it acts as the active context. The possible values are as follows:

- Disable - The default active singleton is not created.
- ReactiveWithoutContext - The default active singleton is created if there are reactive classes that consume events and do not have an active context explicitly specified. The default active singleton can handle only these classes.
- All - The default active singleton is generated if there is at least one event-consuming reactive class and the active singleton can handle all reactive classes that consume events - even those reactive classes that specify another active class as their active context.

Default = ReactiveWithoutContext

DefaultImplementationDirectory

The DefaultImplementationDirectory property specifies the relative path to the default directory for generated implementation files. The value of this property is added after the configuration path. Consider the following case:

- File C.cpp is an implementation of class C mapped to a folder Foo.
- The active configuration (cfg) is under component cmp.
- DefaultImplementationDirectory is set to “src”

Rhapsody generates C.cpp to root>\cmp\cfg\src\Foo. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

Default = Empty string

DefaultSpecificationDirectory

The DefaultSpecificationDirectory property specifies the relative path to the default directory for generated specification files. The value of this property is added after the configuration path. Consider the following case:

- File B.h is a specification of class B that is not mapped to any file.
- The active configuration (cfg) is under component cmp.
- DefaultSpecificationDirectory is set to “inc”

Rhapsody generates B.h to root>\cmp\cfg\inc. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

Default = Empty string

DependencyRuleScheme

The DependencyRuleScheme property specifies how dependency rules should be generated in the makefile. The possible values are as follows:

- Basic - Generates only the local implementation and specification files in the dependency rule in the makefile.
- ByScope - In addition to generating the same files as the Basic option, generates the specification files of related elements (dependencies, associations, generalizations, and so on) that are in the scope of the active component.
- This option corresponds to the Rational Rhapsody 5.0.1 behavior.
- Extended - In addition to generating the same files as the ByScope option, generates the specification files of related external elements (specified using the properties CG::Class/Package::UseAsExternal) and elements that are not in the scope of the active component.

Default = ByScope

DescriptionBeginLine

This property enables you to specify the prefix for the beginning of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation.

This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected.

Default = //

DescriptionEndLine

This property enables you to specify the prefix for the end of comment lines in the generated code. This

functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation. This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected.

EmptyArgumentListName

The EmptyArgumentListName specifies the string generated for the argument list when an operation has no arguments. For example, if you set this value to "void," for an operation foo that has no arguments, Rational Rhapsody generates the following code:

int foo (void){...}

Default = Empty string

Environment

The Environment property determines the target environment for a configuration. Generated code is targeted for that environment. See the Release Notes for the environments supported by Rhapsody “out-of-the-box.”

“Out-of-the-box” support means that Rational Rhapsody includes a set of preconfigured code generation properties for the environment and precompiled versions of the relevant OXF libraries. The precompiled OXF libraries have been fully tested. You can also add new environments, for example if you want to generate code for another RTOS.

This involves retargeting the OS wrapper files in the Rational Rhapsody framework and creating a new set of code generation properties for the target environment.

Default = Microsoft

ExternalGenerationTimeout

The ExternalGenerationTimeout property specifies how long, in seconds, Rational Rhapsody waits for the each class in the configuration scope to complete so you can once again make changes to the model.

This property applies to both the full-featured external generator and makefile generator. For example, if you set this property to 2 and you have 10 classes, Rational Rhapsody sets a timeout of 20.

If the external code generator does not complete generation in this timeframe, Rational Rhapsody displays a message in the output window saying that the generator is not responding, and you are allowed to make changes to the model.

If you set this property to 0, Rational Rhapsody will not time out the generation session, and waits for the code generator to complete its task - even if it takes forever. Rhapsody waits for a notification from the full-featured external code generator, or for the process termination of a makefile generator.

Default = 0

ExternalGeneratorFileMappingRules

The ExternalGeneratorFileMappingRules property specifies whether the external code generator uses the same file mapping and naming scheme (mapping rules) as Rational Rhapsody.

If the mapping rules are different , the external generator must implement handlers to the GetFileName, GetMainFileName, and GetMakefileName events that Rational Rhapsody runs to get a requested file name and path. The possible values are as follows:

- AsRhapsody - The external generator uses the same mapping rules as Rational Rhapsody.
- DefinedByGenerator - The external generator has its own mapping rules.

Default = AsRhapsody

GenerateAnnotationsForNonSPARKConfigurations

The GenerateAnnotationsForNonSPARKConfigurations property specifies whether

Default = False

GenerateDirectoryPerModelComponent

The GenerateDirectoryPerModelComponent property specifies whether to generate a separate directory for each package in the component. The possible values are as follows:

- True - Rational Rhapsody creates a separate directory for each package in the component.
- False - A separate directory is not created for each package.

Default = True

GeneratorExtraPropertyFiles

The GeneratorExtraPropertyFiles property launches the default Text Editor allowing the user to edit the $OMROOT\CodeGenerator\GenerationRules\LangC\RiC_CG.ini file.

GeneratorRulesSet

The GeneratorRulesSet property enables you to specify your own rules set. Default = Empty MultiLine

GeneratorScenarioName

The GeneratorScenarioName property specifies the scenario name for the rule, if you write your own set of code generation rules. Default = Empty string

GenericEventHandling

The GenericEventHandling property is a Boolean value that determines whether to generate generic event-handling code.

This property supports large-scale collaboration, where you might not be aware of which classes consume a base event of your part in the event hierarchy, and might not have access to parts of the model that use base events.

Beginning with Rational Rhapsody 4.0, the framework base event class includes a new, virtual method that checks the event ID against the specified ID, thereby supplying a generic mechanism for events without super events. The language-specific methods are as follows: C:

#define RiCEvent_isTypeOf(event, id) ((event)-lId == (id)) C++: virtual OMBoolean isTypeOf(short id) const {return lId ==id;}

In addition, C++ includes a new macro, IS_EVENT_TYPE_OF(id), to support both reusable and flat code generation schemes.

Java:

boolean isTypeOf(long id) {return lId == id;}

Each generated event that has a super event will override the method to check the ID against its own ID, then calls its base event directly to continue the check. An event without a base event will return Cleared if the ID does not equal its own.

When you set the GenericEventHandling property to Cleared, event consumption code is generated as in version 3.0.1. Setting this property affects only the way events are consumed - the override on the isTypeOf() method is still generated, to allow handling of events in components that use the generic event handling.

To support complete generic event handling, you should regenerate the code for all events and reactive classes.

Default = Checked

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.

- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

InitializeEmbeddableObjectsByValue

The InitializeEmbeddableObjectsByValue property specifies whether embeddable classes and object types selected in the configuration initial instances list should be allocated by value in the main() routine.

Default = Cleared

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation. Default = Empty MultiLine

MainFunctionArgList

This property provides a list of the main function arguments. The default list is "int argc, char* argv[]."

Default = int argc, char argv[]*

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip.

When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters \n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = None

ShowCgSimplifiedModelPackage

The first step of the code generation process consists of the building of a simplified model based on the Rational Rhapsody model.

By default, the simplified model is not displayed in Rational Rhapsody. To have the simplified model displayed in the browser, set the property ShowCgSimplifiedModelPackage property to True. Once you have done so, the next time you generate code, the simplified model is added automatically at the top of the project tree in the browser.

Default = Cleared

SourceListFile

The SourceListFile property specifies the name of the file containing a list of .java source files to be compiled with javac. The batch file used by the Build command (jdkmake.bat) can use the following call, rather than including a long list of source files: javac –g @files.lst This same command is generated from the following line in the MakeFileContent property for Java: javac –g @$SourceListFile If the SourceListFile property is empty, $SourceListFile is replaced with a string containing all source file names, separated by spaces (for example, "A.java B.java"). This means that if the MakeFileContent default value is not changed, you will get: javac –g @A.java B.java … If you do not want to use the file containing the list of sources, you must also change the MakeFileContent property to replace "javac –g @$SourceListFile" with "javac –g $SourceListFile". Default = files.lst

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

Cygwin

The Cygwin metaclass controls the environment settings (Compiler, framework libraries, etc.) for Cygwin.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/Cygwin

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

AnimIncludeDirectories

The property AnimIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/aom
$(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangCpp/lib/cygwinaomanim$(LIB_EXT)

AnimOxfLibs

The property AnimOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangCpp/lib/cygwinoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/cygwinomcomappl$(LIB_EXT)

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMANIMATOR $(DEFINE_QUALIFIER)__USE_W32_SOCKETS

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = $makefile

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = cygwinmake.bat

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\"\etc\cygwinmake.bat cygwinbuild.mak "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Checked

CompilerFlags

The property CompilerFlags allows you to define additional compilation flags. The value of the property is inserted into the value of the property CompileSwitches (Linux) or CPPCompileSwitches (cygwin). In the generated makefile, you can see the value of this property in the line that begins with ConfigurationCPPCompileSwitches=.

Default = Blank

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

CPPCompileSwitches

The CPPCompileSwitches property specifies the compiler switches.

Default =

$IncludeDirectories $DefinedSymbols $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) $CompilerFlags $OMCPPCompileCommandSet -c

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Checked

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the

property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled).

During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default is as follows:

$(OMROOT)\LangCpp\lib\cygwinWebComponents$(LIB_EXT),
$(OMROOT)\lib\cygwinWebServices$(LIB_POSTFIX)$(LIB_EXT), -lws2_32

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\cygwinrun.bat\" $executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment.

To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\cygwinmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkerFlags

The property LinkerFlags allows you to define linker flags. The value of the property is inserted into the value of the property LinkSwitches. In the generated makefile, you can see the value of this property in the line that begins with LINK_FLAGS=.

Default = Blank

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -O

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet $LinkerFlags

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

The default is as follows: ############## Target type (Debug/Release) ###############

##
CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet
SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll
ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches
$OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True"
ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL"
!ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################### ##
INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib
LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches
LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################## ## $OMContextMacros
OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE
CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile.

The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following:

FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT
CPP_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt
LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel
TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule
TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath
ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows:
################### Predefined macros ################
$(OBJS) : $(INST_LIBS)
$(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF
"$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG
LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library"
LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation"
INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I
$(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS=

OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################## Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NoneIncludeDirectories

The property NoneIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = Blank

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = Blank

NoneOxfLibs

The property NoneOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangCpp/lib/cygwinoxf$(LIB_EXT)

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9:]+)[:] (error|warning)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9:]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (make)[:](.)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (error|undefined|cannot find|multiple definition)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9:]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

*Default = *

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

TraceIncludeDirectories

The property TraceIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/aom
$(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangCpp/lib/cygwintomtrace$(LIB_EXT)
$(OMROOT)/LangCpp/lib/cygwinaomtrace$(LIB_EXT)

TraceOxfLibs

The property TraceOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangCpp/lib/cygwinoxfinst$(LIB_EXT)
$(OMROOT)/LangCpp/lib/cygwinomcomappl$(LIB_EXT)

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMTRACER

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Cleared

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Cleared

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Checked

Dependency

The Dependency metaclass controls the dependency for a package that defines a namespace.

CreateUseStatement

The CreateUseStatement property determines whether a use statement is added to the code after the with statement. The supplier of the dependency must be a class or type. Default = False

GenerateOriginComment

When set to Checked, generates a comment before #include statements indicating which element "caused" the #include.

Default = Checked

GeneratePragmaElaborate

The GeneratePragmaElaborate property determines whether to generate an elaborate pragma for the supplier class in the client class or package. Default = False

GeneratePragmaElaborateAll

The GeneratePragmaElaborateAll property determines whether to generate a pre-elaborate pragma for the supplier class in the client class or package. Default = False

GenerateWithClause

The GenerateWithClause property determines whether with clauses are generated for Usage dependencies.

For example, you can generate a with clause for a package, P1, in the specification of another package, P2, using a dependency, D1, and generate a use clause for P1 in the body of P2. In addition, this functionality is useful for modeling inherited annotations across classes and packages.

Default = True

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

IncludeStyle

The IncludeStyle property controls the style of #include statements. Using this property, you can control

the style of a specific dependency, or the entire configuration/component/project.

To set the style for include files that are synthesized based on associations between model elements (for example, setting the type of some attribute to a class), add a «Usage» dependency between the elements and set this property to the appropriate value. The possible values are as follows:

- Default - Use angle brackets for include statements for external elements, and quotes for include statements for other elements.
- Quotes - Enclose include files in quotation marks. For example: #include "A.h"
- When a compiler encounters an include file in quotes, it searches for the file in both the current directory and the directories specified in the include path. Note that the specific algorithm used is compiler-dependent.
- AngledBrackets - Enclose include files in angle brackets. For example: #include A.h
- When a compiler encounters an include file in angle brackets, it searches for the file only in the directories specified in the include path.
- If you set the property to AngledBrackets at the configuration level, you must also change the CG::File::IncludeScheme property to RelativeToConfiguration to ensure successful compilation.

Default = Default

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip.

When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually.

The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element’s annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name.

Some model information (for example, property settings) might be lost.

Default = None

NamespaceAlias

The property NamespaceAlias allows you to take advantage of the C++ namespace alias feature.

The value of the property should be the string you would like to use as the alias for the namespace of the package that the element is dependent upon.

For example, if you have specified a dependency on the nested namespace Hardware, as defined below:

```
namespace Equipment

{

namespace Hardware

{

class Printer { ... };

}

}
```

you can enter hw for the value of the property NamespaceAlias, and then the generated code will include the following statement:

```
namespace hw = Equipment::Hardware;
```

Because namespace aliases are an alternative to the use of "using" directives, Rational Rhapsody will ignore the value of the boolean property UseNameSpace if you have entered a value for the property NamespaceAlias for the same dependency.

Default = Blank

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.

- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

UseNameSpace

The UseNameSpace property enables you to model namespace usage. When you set a dependency to a package that defines a namespace and set this property to Checked, Rational Rhapsody generates a "using namespace" statement to the package namespace. Default = Cleared

Event

The Event metaclass contains properties that control events.

AnimInstanceCreate

The AnimInstanceCreate property affects event creation. If you set the C_CG::Event::NoDynamicAllocAnimCreate property to False, Rational Rhapsody does not generate the event creation method, effectively disabling the ability to inject the event in animation. To enable the injection of the event, you can specify a different method to obtain an instance of the event by setting this property to the name of the method to use.

Default = Empty string

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property

enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to “MYDLL_API.” The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class.

Default = Empty string

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description

Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

$Tag - The value of the specified element’s tag

$Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

Keyword names can be written in parentheses. For example: If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- Checked - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for

the object or object type.

- Cleared - Events are dynamically allocated during initialization, but not during run time. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during run time.

If you are managing your own memory pools, set this property to False and call CPPReactive_gen() directly. The following example shows how to call RiCReactive_gen() directly to send a static event to a reactive object A, when using a member function of A genStaticEv2A():

void A_genStaticEv2A(struct A_t* const me) { { /*#[operation genStaticEv2A() */ static struct ev _ev; ev_Init(_ev); RiCEvent_setDeleteAfterConsume(((RiCEvent*)_ev), RiCFALSE); (void) RiCReactive_gen(me-ric_reactive, ((RiCEvent*)_ev), RiCFALSE); /*#]*/ } }

Alternatively, you can use internal memory pools by setting the property BaseNumberOfInstances, which results in the use of framework memory pools. If you use the framework memory pools, do not disable the Create() and Destroy() methods because these methods are used to manage the memory pool.

When you disable the generation of the Create() and Destroy() methods, you can still inject events in animation by supplying an alternate function to get an event instance. To do this, set the AnimInstanceCreate property.

Default = Checked

In

The property In determines the exact syntax used when an event is used as an "in" parameter for an operation.

Default = const $type&

InOut

The property InOut determines the exact syntax used when an event is used as an "in/out" parameter for an operation.

Default = $type&

Out

The property Out determines the exact syntax used when an event is used as an "out" parameter for an operation.

Default = $type&*

ReturnType

The property ReturnType determines the exact syntax used when an event is used as the return type of an operation.

*Default = $type**

File

The File metaclass contains properties that control the generated code files.

DiffDelimiter

The DiffDelimiter property defines a symbol that is used to avoid overwriting an unchanged line of code during code generation. Use this property to avoid touching the source code file when the "diff-delimited" line has not changed. In general, fewer source files need to be recompiled if fewer source files are touched. For example, the DiffDelimiter symbol “//!” is used in the CPP_CG::File::Header property. This symbol is at the beginning of a line of code that includes the current code generation date. The code generator compares the code it would normally generate for that line (the current code generation date) to that previously generated (the last code generation date). If the date has not changed, the line is not overwritten, possibly preventing the file’s modification time from changing (being "touched").

Default = //!

Footer

The Footer property specifies a multiline footer that is added to the end of generated Java files.

Default =

"/********** File Path: $FullCodeGeneratedFileName **********/"

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.

- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

Header

The Header property specifies a multiline header that is added to the top of all generated Java files.

Default =

/********** Rhapsody : $RhapsodyVersion Login : $Login Component : $ComponentName Configuration : $ConfigurationName Model Element : $FullModelElementName //! Generated Date : $CodeGeneratedDate File Path : $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with

_DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationFooter

The ImplementationFooter property specifies the multiline footer to be generated at the end of implementation files. The default footer template for C++ is as follows:

/********** File Path: $FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords
- Property keywords

- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

ImplementationHeader

The ImplementationHeader property specifies the multiline header that is generated at the beginning of implementation files. The default header template for C++ is as follows:

/********** Rhapsody: $RhapsodyVersion Login: $Login Component: $ComponentName Configuration: $ConfigurationName Model Element: $FullModelElementName //! Generated Date: $CodeGeneratedDate File Path: $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)

- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip.

When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually.

The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between

the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. Default = Auto

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationFooter

The SpecificationFooter property specifies the multiline footer to be generated at the end of specification files.

The default is as follows:

/********** File Path: $FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

SpecificationHeader

The SpecificationHeader property specifies the multiline header to be generated at the beginning of specification files.

The default is as follows:

/********** Rhapsody: $RhapsodyVersion Login: $Login Component: $ComponentName Configuration: $ConfigurationName Model Element: $FullModelElementName //! Generated Date: $CodeGeneratedDate File Path: $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter

value is “//!”. The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class).

For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to "abstract." You must include the space after the word "abstract." If the visibility for the class is set to default, the following class declaration is generated in the .java file:

abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class Yes No Inside Package Yes Yes Inside

Default = Empty MultiLine

Framework

The Framework metaclass contains properties that affect the Rational Rhapsody framework.

ActivateFrameworkDefaultEventLoop

The ActivateFrameworkDefaultEventLoop property specifies the framework call that initializes the framework main event loop.

Default = OXF::start($Fork);

The value of $Fork is calculated from the property CG::Configuration::StartFrameworkInMainThread for regular applications and from the property CORBA::Configuration::StartFrameworkInMainThread for CORBA servers.

This property can be set at the configuration level or higher.

ActiveBase

The ActiveBase property specifies the superclass from which to specialize all threads, if the ActiveBaseUsage property is set to Checked.

Default = OMThread

ActiveBaseUsage

The ActiveBaseUsage property specifies whether to use the superclass specified by the ActiveBase property as the superclass for all threads.

Default = Checked

ActiveDestructorGuard

The ActiveDestructorGuard property specifies the macro that starts protection for an active user object destructor.

Default = START_DTOR_THREAD_GUARDED_SECTION

ActiveExecuteOperationName

The ActiveExecuteOperationName property sets the user object virtual table for an active object and passes it to a task in the task initialization function (RiCTask_init()). Follow these steps:

- Create a method with the following signature: struct RiCReactive * operation name> (RiCTask * const)
- Set the operation name in the ActiveExecuteOperationName property.
- Start the execution of the active object task by calling the RICTASK_START() macro on the object.

The virtual function table member name is stored in the ActiveVtblName property.

Default = Empty string

ActiveGuardInitialization

The ActiveGuardInitialization property specifies the call that makes the active object event dispatching guarded.

Default = setToGuardThread

ActiveIncludeFiles

The ActiveIncludeFiles property specifies the base class for threads when using selective framework includes. If a class is active and this property is defined, the file specified by the property is included in the class specification file.

Default = oxf/omthread.h

ActiveInit

The ActiveInit property specifies the format of the declaration generated for the initializer for an active class.

(Default =

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active objects, if the ActiveMessageQueueSize property for classes is left blank.

Default = OMOSThread::DefaultMessageQueueSize

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects, if the ActiveStackSize property for classes is left blank.

Default = is OMOSThread::DefaultStackSize

ActiveThreadName

The ActiveThreadName property specifies the name of threads, if the ActiveThreadName property for classes is left blank.

Default = ""

ActiveThreadPriority

The ActiveThreadPriority priority specifies the priority of threads, if the ActiveThreadPriority property for classes is left blank.

Default = OMOSThread::DefaultThreadPriority

ActiveVtblName

The ActiveVtblName property stores the name of the virtual function table associated with a task (the RiCTask member of the structure). Default = $objectName_activeVtbl

BooleanType

The BooleanType property specifies the Boolean type used by the framework. Default = RhpBoolean

CurrentEventId

The CurrentEventId property specifies the call or macro used to obtain the ID of the currently consumed event. Default = OM_CURRENT_EVENT_ID

DefaultProvidedInterfaceName

The DefaultProvidedInterfaceName property specifies the interface that must be implemented by the "in" part of a rapid port. See the Rational Rhapsody Help for more information on rapid ports. Default = DefaultProvidedInterface

DefaultReactivePortBase

The DefaultReactivePortBase property stores the base class for the generic rapid port (or default reactive port). This base class relays all events. See the Rational Rhapsody Help for more information on rapid ports. Default = OMDefaultReactivePort

DefaultReactivePortIncludeFiles

The DefaultReactivePortIncludeFiles property specifies the include files that are referenced in the generated file that implements the class with the rapid ports. See the Rational Rhapsody Help for more information on rapid ports.

Default = <oxf/OMDefaultReactivePort.h>

DefaultRequiredInterfaceName

The DefaultRequiredInterfaceName property specifies the interface that must be implemented by the "out" part of a rapid port. See the Rational Rhapsody Help for more information on rapid ports.

Default = DefaultRequiredInterface

EnableDirectReactiveDeletion

The EnableDirectReactiveDeletion property specifies the call to the framework that supports direct deletion of reactive instances (using the delete operator) instead of graceful framework termination (using the reactive destroy() method).

When using destroy(), the object waits in a zombie mode until all the events that are designated to it are removed from the active context queue, and then self -destructs.

In this scheme, there is no need to traverse the queue of the active context to cancel pending events, and there is no need to make the reactive destructor guarded to ensure safe deletion. A reactive object can be either in a graceful termination or forced deletion (using the delete operator) state: you cannot use graceful deletion on an object that allows forced deletion, and vice versa.

You can set a single reactive object in a forced deletion state, or set the entire system (all reactive instances) in a forced deletion state (as is done for backward compatibility). Graceful termination should not be used when a reactive part (of a composite class) runs in a context of an active object that is not part of, and different from, the composite active context.

If you are using a Rhapsody library component as part of an application where the main is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init(). If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call. Note that the property CPP_CG::Framework::UseDirectReactiveDeletion must be set to True for this property to take effect. When it is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init().

Default = OXF::supportExplicitReactiveDeletion();

EventBase

The EventBase property specifies the base class for all events, if the EventBaseUsage property is set to Checked.

Default = OMEvent

EventBaseUsage

The EventBaseUsage property specifies whether to use the event superclass specified by the EventBase property as the parent of all events.

Default = Checked

EventGenerationPattern

The EventGenerationPattern property supplies some of the information needed to generate code for Send Action elements in C, C++, and Java.

For each language, code generation for this element is determined by the following properties:

- CPP_CG::Framework::EventGenerationPattern - general format
- CPP_CG::Framework::EventToPortGenerationPattern - used when sending even to a port

Note: Rhapsody does not support roundtripping for Send Action elements.

EventIncludeFiles

The EventIncludeFiles property specifies the base class for events when using selective framework includes. If events are defined in a package, the file specified by this property is included in the package specification file to enable the use of events and timeouts in the package.

Default = <oxf/event.h>

EventSender

The property EventSender specifies the base class to use for reactive interfaces when the property ReactiveInterfaceScheme is set to Thin. This allows you to define your own interface for event sending behavior instead of the framework class IOxfEventSender.

(Default = IOxfEventSender)

EventSetParamsStatement

The EventSetParamsStatement property specifies a template for the body of the setParams() method, provided by the Rational Rhapsody framework for Java, to set the parameters of an event. For example, for an event of type evOn(), the default template would generate the following code in the body of the setParams() method: evOn params = (evOn) event; The default value is as follows: $eventType params = ($eventType) event;

FrameworkInitialization

The FrameworkInitialization property specifies the framework initialization code that is called by the main.

The default is as follows:

OXF::initialize($(Argc)$(Argv)$(AnimationPortNumber)$(RemoteHost)$(TimerResolution)$(TimerMaxTimeouts)$(TimeM

HeaderFile

The HeaderFile property specifies the framework header files to be included in objects that are within the

scope of a particular configuration.

Default = oxf/oxf.h

IncludeHeaderFile

The IncludeHeaderFile property specifies whether to include the framework header files specified by the CG::Framework::HeaderFile property in the project.

Default = Checked

InnerReactiveClassName

The InnerReactiveInstanceName property enables you to specify the name of a reactive class that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive.

The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance.

Default = Reactive

InnerReactiveInstanceName

The InnerReactiveInstanceName property enables you to specify the name of a reactive instance that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive.

The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance.

Default = reactive

InstrumentVtblName

The InstrumentVtblName property specifies the name of the virtual function table associated with animation objects. Each animated object has its own virtual function table (Vtbl). This table enables you to create your own framework, with its own virtual instrumentation functions, and connect it to Rational Rhapsody.

Default = $objectName_instrumentVtbl

IsCompletedCall

The IsCompletedCall property specifies the call or macro that determines whether the state reached a final state so it can be exited on a null transition. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name. Default = IS_COMPLETED($State)

IsInCall

The IsInCall property specifies the query that determines whether the state is in the current active configuration. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name. Default = IS_IN($State)

MakeFileName

The MakeFileName property enables you to specify a new name for the makefile. To use this property, add the following line to the .prp file: Property MakeFileName String "MyFileName"

In this syntax, MyFileName specifies the name of the makefile.

NullTransitionId

The NullTransitionId property specifies the ID reserved for null transition consumption. Default = OMNullEventId

OperationGuard

The OperationGuard property specifies the macro that guards an operation. Default = GUARD_OPERATION

ProtectedBase

The ProtectedBase property specifies the base class for protected objects, if the ProtectedBaseUsage property is set to Checked.

Default = Empty string

ProtectedBaseUsage

The ProtectedBaseUsage property specifies whether to use the class specified by the ProtectedBase property as the base class for protected objects.

Default = Checked

ProtectedClassDeclaration

The ProtectedClassDeclaration property affects how protected classes are implemented. Beginning with Rational Rhapsody 4.0, instead of inheriting from OMProtected, the class embeds an aggregate OMProtected. The aggregate member and helper methods are defined in the macro OMDECLARE_GUARDED (defined in omprotected.h). Default = OMDECLARE_GUARDED

ProtectedIncludeFiles

The ProtectedIncludeFiles property specifies the base class for protected classes when using selective framework includes.

Default value = <oxf/omprotected.h>

ProtectedInit

The ProtectedInit property specifies the declaration generated for the initializer for guarded objects. The default value for Ada is an empty string. The default value for C is as follows: $base_init($member)

ReactiveBase

The ReactiveBase property specifies the base class for all reactive classes, if the ReactiveBaseUsage property is set to Checked.

Default = OMReactive

ReactiveBaseUsage

The ReactiveBaseUsage property specifies whether to use the class specified by the ReactiveBase property as the base class for all reactive objects.

Default = Checked

ReactiveConsumeEventOperationName

The ReactiveConsumeEventOperationName property sets the user object virtual table for a reactive object. Follow these steps:

- Create a method with the following signature: void operation name>(RiCReactive * const, RiCEvent*)
- Set the operation name in the ReactiveConsumeEventOperationName property.

Rational Rhapsody Developer for Ada ignores all the values for the properties under the Framework metaclass except for this one. Default = Empty string

ReactiveCtorActiveArgDefaultValue

The ReactiveCtorActiveArgDefaultValue property specifies the default value of the active context argument in a reactive constructor. Default = 0

ReactiveCtorActiveArgName

The ReactiveCtorActiveArgDefaultValue property specifies the name of the active context argument in a reactive constructor. Default = theActiveContext

ReactiveCtorActiveArgType

*The ReactiveCtorActiveArgDefaultValue property specifies the type of the active context argument in a reactive constructor. Default = IOxfActive**

ReactiveDestructorGuard

The ReactiveDestructorGuard property specifies the macro that starts protection of a section of code used for destruction of a reactive instance. This prevents a "race" (between the deletion and event dispatching) when deleting an active instance. Default = START_DTOR_REACTIVE_GUARDED_SECTION

ReactiveEnableAccessEventData

The ReactiveEnableAccessEventData property specifies the code to be used to enable access to the specific event data in a transition (typically by assigning a local variable of the appropriate type). The property supports the $Event keyword so you can specify the event type. Default = OMSETPARAMS($Event);

ReactiveGuardInitialization

The ReactiveDestructorGuard property specifies the framework call that makes the event consumption of a specific reactive class guarded. Default = setToGuardReactive

ReactiveHandleEventNotConsumed

The ReactiveHandleEventNotConsumed property registers a method to handle unconsumed events in a reactive class. Specify the method name as this property's value. Default = Empty string

ReactiveHandleTONotConsumed

The ReactiveHandleTONotConsumed property registers a method to handle unconsumed trigger operations in a reactive class. Specify the method name as this property's value. Default = Empty string

ReactiveIncludeFiles

The ReactiveIncludeFiles property specifies the base classes for reactive classes when using selective framework includes. If a class is reactive and this property is defined, the file specified by the property is included in the class specification file. For reactive classes, the header files specified by the following properties are also included:

- EventIncludeFiles - For the event base class
- ActiveIncludeFiles - If the class is guarded or instrumented

Default = <oxf/omreactive.h,oxf/state.h>

ReactiveInit

The ReactiveInit property specifies the declaration for the initializer generated for reactive objects. The default pattern for C is as follows: $base_init($member, (void*)$mePtr, $task, $VtblName); The $base variable is replaced with the name of the reactive object during code generation. The string "_init" is appended to the object name in the name of the operation. For example, if the reactive object is named A, the initializer generated for A is named A_init(). The $member variable is replaced with the name of the reactive member (equivalent to the base class) of the object during code generation.

The $mePtr variable is replaced with the name of the user object (the value of the Me property). The member and mePtr objects are not equivalent if the user object is active. The $VtblName variable is replaced with the name of the virtual function table for an object, specified by the ReactiveVtblName property. The default value for Ada is an empty string. The default for C is as follows:

$base_init($member, (void*)$mePtr, $task, $VtblName);

The default for Java is as follows: reactive = new Reactive($task);

ReactiveInterface

The ReactiveInterface property specifies the name of the interface class that forwards messages to an inner class instance of a reactive class in order to implement its reactive behavior. Default = RiJStateConcept

ReactiveSetEventHandlingGuard

The ReactiveSetEventHandlingGuard property enables you to control the code generated within the constructor of a reactive class. When you use this property with guarded triggered operations, it enables guarding of the event handling (in order to provide mutual exclusion between the event and TO handling). Default = setEventGuard(getGuard());

ReactiveSetTask

The ReactiveSetTask property specifies the string that tells a reactive object whether it is an active or a sequential instance.

Default = setActiveContext($task, $isActive);

ReactiveStateType

The ReactiveStateType property is used for serialization to define the oxfstate type.

Default = unsigned long

ReactiveVtblName

The ReactiveVtblName property specifies the name of the virtual function table (Vtbl) associated with a reactive object. Each reactive object has its own Vtbl, which enables you to create your own framework and connect it to Rational Rhapsody. Default = $objectName_reactiveVtbl

SetManagedTimeoutCanceling

The SetManagedTimeoutCanceling property is a property for backward compatibility that specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (where OMTimerManager is responsible for cancellation of timeouts) or the Rational Rhapsody 6.0 scheme.

In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object). This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance).

The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling).

If you are using a Rhapsody library component as part of an application where the main is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init().

If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call.

Default = OXF::setManagedTimeoutCanceling(true);

SetRhp5CompatibilityAPI

The SetRhp5CompatibilityAPI property specifies the call that configures models created before Rhapsody 6.0 so they use the 5. x version of the framework instead of the new one. See UseRhp5CompatibilityAPI for more information on Version 5. x compatibility mode. Default = OXF::setRhp5CompatibleAPI(true);

StaticMemoryIncludeFiles

The StaticMemoryIncludeFiles property specifies the files to be included in the package specification file

if static memory management is enabled and you are using selective framework includes.

Default = <oxf/MemAlloc.h>

StaticMemoryPoolDeclaration

The StaticMemoryPoolDeclaration property specifies the declaration of the memory pool for timeouts.

Default = DECLARE_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances)

StaticMemoryPoolImplementation

The StaticMemoryPoolImplementation property specifies the generated code in the implementation file for a memory pool implementation (see the BaseNumberOfInstances property).

Default = IMPLEMENT_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances, $AdditionalNumberOfInstances, $ProtectStaticMemoryPool)

TestEventTypeCall

The TestEventTypeCall property specifies the test used in event consumption code to check if the currently consumed event is of a given type. Default = IS_EVENT_TYPE_OF($Id)

TimeoutId

The TimeoutId property specifies the ID reserved for timeout events. Default = OMTimeoutEventId

TimerMaxTimeouts

The TimerMaxTimeouts property specifies the maximum number of timeouts allowed simultaneously in the system, if the TimerMaxTimeouts property for the configuration is not overridden. In the framework, the default number of timers is 100. Default = Empty string

TimerResolution

The property TimerResolution allows you to override the default tick time used.

The number entered is the number of milliseconds used for the tick time.

The default tick time (currently 100 milliseconds) is defined by OMTimerManagerDefaults::defaultTicktime in the file OMTimerManagerDefaults.cpp

Default = Blank

UseDirectReactiveDeletion

The UseDirectReactiveDeletion property determines whether direct deletion of reactive instances (using the delete operator) is used instead of graceful framework termination (using the reactive destroy() method). When this property is set to Checked, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init(). See EnableDirectReactiveDeletion and the upgrade history on the support site for more information on this functionality. Default = Cleared

UseManagedTimeoutCanceling

The UseManagedTimeoutCanceling property specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (so OMTimerManager is responsible for cancellation of timeouts). In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object).

This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling). When loading a pre-6.0 model, Rational Rhapsody sets the project CPP_CG::Framework::UseManagedTimeoutCanceling to Checked to set the system-compatibility mode.

See the upgrade history on the support site for more information.

Default = Cleared

UseRhp5CompatibilityAPI

The UseRhp5CompatibilityAPI property specifies whether to use the virtual functions of the core implementation classes that existed in the pre-Rhapsody 6.0 framework. The Rhapsody 6.0 framework introduces a set of interfaces for the core behavioral framework. The interfaces define a concise API for the framework and enable you to replace the actual implementation of these interfaces while maintaining the framework behavior.

As a result of the interfaces' introduction, the framework behavioral classes (OMReactive, OMThread, and OMEvent) use a new set of virtual operations to implement the interfaces and provide the behavioral infrastructure. To support existing customizations of these classes (made by inheriting and overriding the virtual operations), the framework can work in a mode where the pre-6.0 API virtual operations are called.

When loading a pre-6.0 model, Rational Rhapsody sets the project property CPP_CG::Framework::UseRhp5CompatibilityAPI to True to set the system-compatibility mode. If this is set to Checked, the pre-6.0 API is called by the framework instead of the interface-based API. Without this flag, user customizations will compile but will not be called. See the upgrade history on the support site for more information on the Version 5. x compatibility mode.

Default = Cleared

General

The General metaclass contains a property that specifies the Friend implementation scheme.

<<Friend>>ImplementationScheme

The <<Friend>>ImplementationScheme property specifies how a friendship relation between classes is generated into code. According to the UML version 1.3 standard, a dependency that is stereotyped as <<Friend>> from a class A to a class B means that A is a friend of B.

Therefore, in the C++ implementation, class B grants friendship to class A. Rhapsody 3.0 and higher supports both semantics. Select the desired semantics by setting the <<Friend>>ImplementationScheme property to the appropriate value:

- UML1.3 - Use the UML <<Friend>> implementation semantics.
- Rhapsody2.3 - Use the Rational Rhapsody 2.3 <<Friend>> implementation semantics.

When using the UML1.3 scheme, you can set the UsageType property so #include macros are generated in the friend's source file. By default, this property is set to Specification, meaning that the #include is generated in the specification file. The default value of the <<Friend>>ImplementationScheme property UML1.3.

However, when you first load a version 2.3 model into Rational Rhapsody version 3.0 or higher, this property is set to Rational Rhapsody 2.3 at the project level. This enables the implementation scheme to be consistent for the entire model.

You can override this property at the package level, so some packages use the UML1.3 scheme whereas others use the Rational Rhapsody V2.3 scheme. Packages that had been previously imported into the model will not return to their old property value. Instead, they receive the project value, unless the property was overridden before the package was imported.

Default = UML1.3

Generalization

The Generalization metaclass contains a property used to support generalization. See the Rational Rhapsody Help for more information on generalization.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to Cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to Cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to Cleared, all the arguments are not animated.
- If the AnimateArguments property is set to Cleared, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = Checked

INTEGRITY

The INTEGRITY metaclass contains the environment settings (Compiler, framework libraries, etc.) for INTEGRITY 4.0.X .

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/INTEGRITY.

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags.

Default = Empty string

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

:optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550

BLDIncludeAdditionalBLD

The IncludeAdditional property enables you to specify additional build options.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model. The default values for the C++ INTEGRITY metaclass are as follows:

:defines=_DEBUG :target_os=integrity :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default =

:defines=_DEBUG :target_os=integrity :integrity_option=dynamic :staticlink=true

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = mbx800

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings

tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" IntegrityBuild.bat buildLibs $BLDTarget bld "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected

format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .mod)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

Default =Integrity

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation,

these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = Empty string

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored.

Default = $OMROOT/etc/IntegrityMakefileGenerator.bat

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .bld)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = work

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. The default values are as follows:

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (error|catastrophic error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in

double quotes in the generated makefile.

Default = Cleared

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = OM_REUSABLE_STATECHART_IMPLEMENTATION.

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors check box (located in the configuration Initialization tab).

Default = Cleared

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Cleared

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

INTEGRITY5

The INTEGRITY5 metaclass contains the environment settings (Compiler, framework libraries, etc.) for INTEGRITY 5.0.X

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/INTEGRITY

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomAnim$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -D_OMINSTRUMENT

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags.

Default = Empty string

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

-I. --diag_suppress 14,550,611,1795 :outputDirRelative=$ObjectsDirectory

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

The default is as follows:

-G-dynamic-non_shared-wantprototype-Ospace-tnone-delete

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default =

-G -non_shared -wantprototype -Ospace -tnone -delete

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = mbx800

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Integrity5Make.bat\" IntegrityBuild.bat buildLibs $BLDTarget "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileDebug

The CompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = Empty string

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property. The default values are as follows:

Default = -DUSE_IOSTREAM

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -D_DEBUG -G

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .mod)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

Default = $OMROOT/MakeTmpl/INTEGRITY5.ld

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

Default = Integrity5

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(LibPrefix)WebComponents$(BLDTarget)$OMLibSuffix$LibExtension ,
$OMRoot/lib/$(FrameworkLibPrefix)WebServices$(BLDTarget)$OMLibSuffix$LibExtension

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = $MULTI_ROOT/rhapsody_multi_ide.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

IntegrityLinkFile

The name of the Integrity link file that should be added to the makefile template.

Default = INTEGRITY5.ld

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = Empty string

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default =

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Integrity5Make.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored.

Default = $OMROOT/etc/Integrity5MakefileGenerator.bat

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LibPrefix

Combines all the prefixes of library names.

Default = $(FrameworkLibPrefix)$(OMMultipleAddressSpacesPrefix)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .gpj)

MakeFileContentForExe1

The content of the makefiles, in case of executable component type.

Default =

#!gbuild primaryTarget=$PrimaryTarget [INTEGRITY Application] -bsp $BLDTarget -os_dir $IntegrityRoot -o $OMTargetName$ExeExtension -object_dir=$ObjectsDirectory $BLDMainExecutableOptions $OMMultipleAddressSpacesLibraries $KernelProject $MakeFileNameForExe2 $IntegrateFile $BSPFile $ConnectionFile $ResourceFile $OMMultipleAddressSpacesAdditionalFiles

MakeFileContentForExe2

The content of the makefiles, in case of executable component type.

Default =

#!gbuild [Program] -o $OMTargetName -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangCpp -L$OMRoot/LangCpp/lib $OMUserIncludePath $LinkSwitches $OMCompilationFlag $CompileSwitches $OMReusableFlag $OMInstrumentationFlags $OMInstrumentationLibs $OMMultipleAddressSpacesLibraries $BLDAdditionalDefines $OMUserLibs $OMMainFiles$ImpExtension $OMSrcFiles $IntegrityLinkFile $LinkerFile

MakeFileContentForLib1

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -bsp $BLDTarget -os_dir $IntegrityRoot -object_dir=$ObjectsDirectory $BLDMainLibraryOptions $OMMultipleAddressSpacesSwitches $KernelProject $MakeFileNameForLib2 $IntegrateFile $BSPFile $ConnectionFile $ResourceFile $OMMultipleAddressSpacesAdditionalFiles

MakeFileContentForLib2

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild [Library] -o $OMTargetName$LibExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangCpp $OMUserIncludePath $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $OMReusableFlag $BLDAdditionalDefines $OMSrcFiles

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForExe2

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)_program$MakeExtension

MakeFileNameForLib1

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForLib2

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)_library$MakeExtension

MultipleAddressSpacesIntFileContent

The content of the MultipleAddressSpacesIntFileName file.

Default =

Integrate configuration file for compiling a list of address spaces that use also POSIX shared memory manager Kernel Filename DynamicDownload EndKernel $OMSubComponentInfo AddressSpace shared_memory_manager Filename posix_shm_manager MaximumPriority 200 Language $OMLanguage Task Initial StartIt true EndTask EndAddressSpace

MultipleAddressSpacesIntFileName

a file with this name is created in case of multiple address space compilation.

Default = $OMTargetName.int

MultipleAddressSpacesLibraries

names of libraries to add in case of multiple address space usage.

Default =

-l$(FrameworkLibPrefix)Dox$(BLDTarget)$LibExtension -llibposix$LibExtension

-llibshm_client$LibExtension

MultipleAddressSpacesPrefix

A prefix that is added to libraries in case of multiple address space compilation. OMMultipleAddressSpacesPrefix keyword will add this prefix when needed.

Default = Distributed

MultipleAddressSpacesSwitches

A switch for multiple address space compilation.The makefile template can add it directly but it is preferred to use the keyword OMMultipleAddressSpacesSwitches – that checks whether this switch should be added.

Default = -DRIC_DISTRIBUTED_SYSTEM

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

The default is as follows:

-llibsocket.a -llibnet.a

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = work

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. The default values are as follows:

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (error|catastrophic error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

PosixSharedMemoryFiles

This list is copied only in case of multiple address space compilation.

Default = $OMROOT/MakeTmpl/posix_shm_manager.gpj,$OMROOT/MakeTmpl/shm_area.c

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

Default = R

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = OM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInstTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)TomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -DOMTRACER

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors check box (located in the configuration Initialization tab).

Default = Cleared

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

For INTEGRITY, OBJECTADA, RAVEN_PPC, and SPARK the default value is Cleared; for the other environments, the default value is Checked.

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

WebInstLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on.

Integrity5ESTL

The Integrity5ESTL metaclass contains the environment settings (Compiler, framework libraries, etc.) for INTEGRITY 5.0.X. Embedded C++ with Templates.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/INTEGRITY

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomAnim$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMANIMATOR $(DEFINE_QUALIFIER)__USE_W32_SOCKETS

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags.

Default = Empty string

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

The default is as follows:

-I. --diag_suppress 14,550,611,1795 --one_instantiation_per_object --ee --eele
--std_cxx_include_directory $(MULTI_ROOT)/eecxx --std_cxx_include_directory
$(MULTI_ROOT)/ansi

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

Default = -G -dynamic -non_shared -wantprototype -Ospace -tlocal -delete

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default = -G -non_shared -wantprototype -Ospace -tlocal -delete

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = mbx800

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Integrity5Make.bat\" IntegrityBuild.bat buildLibs $BLDTarget ESTL "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileDebug

The CompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = Empty string

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property. The default values are as follows:

Default = -DUSE_IOSTREAM -DOM_ESTL

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -D_DEBUG -G

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty MultiLine

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

If applicable, see also the definition of the EntryPointDeclarationModifier property for more information.

EnvironmentVarName

The EnvironmentVarName property specifies the name of the global variable that you must define in order to use the Embedded C++ compiler. It is used by the MultiMakefileGenerator. The value replaces the $EnvironmentVarName value> keyword inside the property value BLDAdditionalOptions.

Default = MULTI_ROOT

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ESTLCompliance

The ESTLCompliance property is a Boolean value that determines whether you are using the Embedded C++ (ESTL) environment and conform to its requirements. In instrumentation mode, the Rational Rhapsody code generator usually creates an OMAnimatedUser Class> friend class for each user-defined class. This class inherits from AOMInstance, if its User Class> does not inherit from another class in the model. This inheritance is virtual and is needed for multiple inheritance support. Because ESTL does not support multiple inheritance (as far as virtual inheritance), the Rational Rhapsody Developer for C++ code generator will not create “virtual” inheritance if ESTLCompliance is set to True. To support ESTL compliance, Rational Rhapsody includes a new check to recognize the following elements of ESTL-noncompliance:

- Multiple inheritance, caused by the user model (several superclasses)
- Multiple inheritance, caused by Rhapsody (an active reactive class is generated with two base classes: OMReactive and OMThread)
- Multiple inheritance, caused by a combination of the following factors:
- An active class containing a superclass
- A reactive class containing a superclass
- Virtual inheritance, declared by the user in the features of the superclass

In these cases, Rational Rhapsody displays the following warning message for each problematic class: "ESTL does not support multiple/virtual inheritance" Note that this check runs only when the ESTLCompliance property is set to Checked.

Default = Checked

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .mod)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

Default = $OMROOT/MakeTmpl/INTEGRITY5.ld

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

Default = Integrity5ESTL

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the

Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(LibPrefix)WebComponents$(BLDTarget)$OMLibSuffix$LibExtension , $OMRoot/lib/$(FrameworkLibPrefix)WebServices$(BLDTarget)$OMLibSuffix$LibExtension

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = $MULTI_ROOT/rhapsody_multi_ide.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

IntegrityLinkFile

The name of the Integrity link file that should be added to the makefile template.

Default = INTEGRITY5.ld

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = Empty string

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Integrity5Make.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored.

Default = $OMROOT/etc/Integrity5MakefileGenerator.bat

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LibPrefix

Combines all the prefixes of library names.

Default = $(FrameworkLibPrefix)$(OMMultipleAddressSpacesPrefix)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty MultiLine

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .gpj)

MakeFileContentForExe1

The content of the makefiles, in case of executable component type.

Default =

#!gbuild primaryTarget=$PrimaryTarget [INTEGRITY Application] -bsp $BLDTarget -os_dir $IntegrityRoot -o $OMTargetName$ExeExtension -object_dir=$ObjectsDirectory $BLDMainExecutableOptions $OMMultipleAddressSpacesLibraries $KernelProject $MakeFileNameForExe2 $BSPFile $ConnectionFile $ResourceFile

MakeFileContentForExe2

The content of the makefiles, in case of executable component type.

Default =

#!gbuild [Program] -o $OMTargetName -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangCpp -L$OMRoot/LangCpp/lib $OMUserIncludePath $LinkSwitches $OMCompilationFlag $CompileSwitches $OMReusableFlag $OMInstrumentationFlags $OMInstrumentationLibs $OMMultipleAddressSpacesLibraries $BLDAdditionalDefines $OMUserLibs $OMMainFiles$ImpExtension $OMSrcFiles $IntegrityLinkFile $LinkerFile

MakeFileContentForLib1

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -bsp $BLDTarget -os_dir $IntegrityRoot -object_dir=$ObjectsDirectory $BLDMainLibraryOptions $MakeFileNameForLib2 $BSPFile $ConnectionFile $ResourceFile

MakeFileContentForLib2

The content of the makefiles, in case of library component type.

Default =

#!gbuild [Library] -o $OMTargetName$LibExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangCpp $OMUserIncludePath $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $OMReusableFlag $BLDAdditionalDefines $OMSrcFiles

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForExe2

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)_program$MakeExtension

MakeFileNameForLib1

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForLib2

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)_library$MakeExtension

MultipleAddressSpacesIntFileContent

The content of the MultipleAddressSpacesIntFileName file.

Default =

Integrate configuration file for compiling a list of address spaces that use also POSIX shared memory manager Kernel Filename DynamicDownload EndKernel $OMSubComponentInfo AddressSpace shared_memory_manager Filename posix_shm_manager MaximumPriority 200 Language $OMLanguage Task Initial StartIt true EndTask EndAddressSpace

MultipleAddressSpacesIntFileName

A file with this name is created in case of multiple address space compilation.

Default = $OMTargetName.int

MultipleAddressSpacesLibraries

Names of libraries to add in case of multiple address space usage.

Default = -l$(FrameworkLibPrefix)Dox$(BLDTarget)$LibExtension -llibposix$LibExtension -llibshm_client$LibExtension

MultipleAddressSpacesPrefix

A prefix that is added to libraries in case of multiple address space compilation. OMMultipleAddressSpacesPrefix keyword will add this prefix when needed.

Default = Distributed

MultipleAddressSpacesSwitches

A switch for multiple address space compilation.The makefile template can add it directly but it is preferred to use the keyword OMMultipleAddressSpacesSwitches – that checks whether this switch should be added.

Default = -DRIC_DISTRIBUTED_SYSTEM

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

Default = -llibsocket.a -llibnet.a

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = work

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (error|catastrophic error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

PosixSharedMemoryFiles

This list is copied only in case of multiple address space compilation.

Default = $OMROOT/MakeTmpl/posix_shm_manager.gpj,$OMROOT/MakeTmpl/shm_area.c

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

Default = R

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine

running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = OM_REUSABLE_STATECHART_IMPLEMENTATION.

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInstTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)TomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -DOMTRACER

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is

performed after code generation.

Default = Checked

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors check box (located in the configuration Initialization tab).

Default = Cleared

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

For INTEGRITY, OBJECTADA, RAVEN_PPC, and SPARK the default value is Cleared; for the other environments, the default value is Checked.

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

WebInstLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on.

IntegrityESTL

The IntegrityESTL metaclass contains the environment settings (Compiler, framework libraries, etc.) for INTEGRITY 4.0.X. Embedded C++ with Templates.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/INTEGRITY

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags.

Default = OM_ESTL

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

:optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 :cx_mode=extended_embedded :cx_lib=eece :stdcxxincdirs=$(MULTI_ROOT)\eecxx :stdcxxincdirs=$(MULTI_ROOT)\ansi

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

The default is as follows:

:defines=_DEBUG :target_os=integrity :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default =

:defines=_DEBUG :target_os=integrity :staticlink=true

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = mbx800

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings

tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" IntegrityBuild.bat buildLibs bld $BLDTarget ESTL"

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileDebug

The CompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property. The default values are as follows:

Default = -DUSE_IOSTREAM

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -D_DEBUG -G

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

EnvironmentVarName

The EnvironmentVarName property specifies the name of the global variable that you must define in order to use the Embedded C++ compiler. It is used by the MultiMakefileGenerator. The value replaces the $EnvironmentVarName value> keyword inside the property value BLDAdditionalOptions.

Default = MULTI_ROOT

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression

- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ESTLCompliance

The ESTLCompliance property is a Boolean value that determines whether you are using the Embedded C++ (ESTL) environment and conform to its requirements. In instrumentation mode, the Rational Rhapsody code generator usually creates an OMAnimatedUser Class> friend class for each user-defined class. This class inherits from AOMInstance, if its User Class> does not inherit from another class in the model. This inheritance is virtual and is needed for multiple inheritance support. Because ESTL does not support multiple inheritance (as far as virtual inheritance), the Rational Rhapsody Developer for C++ code generator will not create “virtual” inheritance if ESTLCompliance is set to True. To support ESTL compliance, Rational Rhapsody includes a new check to recognize the following elements of ESTL-noncompliance:

- Multiple inheritance, caused by the user model (several superclasses)
- Multiple inheritance, caused by Rhapsody (an active reactive class is generated with two base classes: OMReactive and OMThread)
- Multiple inheritance, caused by a combination of the following factors:
- An active class containing a superclass
- A reactive class containing a superclass
- Virtual inheritance, declared by the user in the features of the superclass

In these cases, Rational Rhapsody displays the following warning message for each problematic class: "ESTL does not support multiple/virtual inheritance" Note that this check runs only when the ESTLCompliance property is set to Checked.

Default = Checked

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .mod)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

The default value = $OMROOT/MakeTmpl/INTEGRITY5.ld

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

Default = IntegrityESTL

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMRoot)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMRoot)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

IntegrityLinkFile

The name of the Integrity link file that should be added to the makefile template.

Default = INTEGRITY5.ld

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file. The default values are as follows:

Default = Empty string

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Integrity5Make.bat\" $makefile $maketarg

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect,

Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored.

Default = $OMROOT/etc/Integrity5MakefileGenerator.bat

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LibPrefix

Combines all the prefixes of library names.

Default = $(FrameworkLibPrefix)$(OMMultipleAddressSpacesPrefix)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .bld)

MakeFileContentForExe1

The content of the makefiles, in case of executable component type.

Default =

#!gbuild primaryTarget=$PrimaryTarget [INTEGRITY Application] -bsp $BLDTarget -os_dir $IntegrityRoot -o $OMTargetName$ExeExtension -object_dir=$ObjectsDirectory $BLDMainExecutableOptions $OMMultipleAddressSpacesLibraries $KernelProject $MakeFileNameForExe2 $IntegrateFile $BSPFile $ConnectionFile $ResourceFile $OMMultipleAddressSpacesAdditionalFiles

MakeFileContentForExe2

The content of the makefiles, in case of executable component type.

Default =

#!gbuild [Program] -o $OMTargetName -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangCpp -L$OMRoot/LangCpp/lib $OMUserIncludePath $LinkSwitches $OMCompilationFlag $CompileSwitches $OMReusableFlag $OMInstrumentationFlags $OMInstrumentationLibs $OMMultipleAddressSpacesLibraries $BLDAdditionalDefines $OMUserLibs $OMMainFiles$ImpExtension $OMSrcFiles $IntegrityLinkFile $LinkerFile

MakeFileContentForLib1

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -bsp $BLDTarget -os_dir $IntegrityRoot -object_dir=$ObjectsDirectory $BLDMainLibraryOptions $OMMultipleAddressSpacesSwitches

$KernelProject $MakeFileNameForLib2 $IntegrateFile $BSPFile $ConnectionFile $ResourceFile $OMMultipleAddressSpacesAdditionalFiles

MakeFileContentForLib2

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild [Library] -o $OMTargetName$LibExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangCpp $OMUserIncludePath $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $OMReusableFlag $BLDAdditionalDefines $OMSrcFiles

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForExe2

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)_program$MakeExtension

MakeFileNameForLib1

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForLib2

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)_library$MakeExtension

MultipleAddressSpacesIntFileContent

The content of the MultipleAddressSpacesIntFileName file.

Default =

Integrate configuration file for compiling a list of address spaces that use also POSIX shared memory manager Kernel Filename DynamicDownload EndKernel $OMSubComponentInfo AddressSpace shared_memory_manager Filename posix_shm_manager MaximumPriority 200 Language $OMLanguage Task Initial StartIt true EndTask EndAddressSpace

MultipleAddressSpacesIntFileName

a file with this name is created in case of multiple address space compilation.

Default = $OMTargetName.int

MultipleAddressSpacesLibraries

names of libraries to add in case of multiple address space usage.

Default =

-l$(FrameworkLibPrefix)Dox$(BLDTarget)$LibExtension -llibposix$LibExtension -llibshm_client$LibExtension

MultipleAddressSpacesPrefix

A prefix that is added to libraries in case of multiple address space compilation. OMMultipleAddressSpacesPrefix keyword will add this prefix when needed.

Default = Distributed

MultipleAddressSpacesSwitches

A switch for multiple address space compilation.The makefile template can add it directly but it is preferred to use the keyword OMMultipleAddressSpacesSwitches – that checks whether this switch should be added.

Default = -DRIC_DISTRIBUTED_SYSTEM

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

The default is as follows:

-llibsocket.a -llibnet.a

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = work

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. The default values are as follows:

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (error|catastrophic error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

PosixSharedMemoryFiles

This list is copied only in case of multiple address space compilation.

Default = $OMROOT/MakeTmpl/posix_shm_manager.gpj,$OMROOT/MakeTmpl/shm_area.c

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

Default = R

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = OM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as

Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors check box (located in the configuration Initialization tab).

Default = Cleared

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

For INTEGRITY, OBJECTADA, RAVEN_PPC, and SPARK the default value is Cleared; for the other environments, the default value is Checked.

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this

keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

WebInstLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on.

Linux

The Linux metaclass controls the environment settings (Compiler, framework libraries, etc.) for Linux.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/Linux

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

AnimIncludeDirectories

The property AnimIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangCpp/lib/linuxaomanim$(LIB_EXT)

AnimOxfLibs

The property AnimOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangCpp/lib/linuxoxfinst$(LIB_EXT)
$(OMROOT)/LangCpp/lib/linuxomcomappl$(LIB_EXT)

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMANIMATOR

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.

- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = $OMROOT/etc/linuxmake linuxbuild.mak build

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompilerFlags

The property CompilerFlags allows you to define additional compilation flags. The value of the property is inserted into the value of the property CompileSwitches (Linux) or CPPCompileSwitches (cygwin). In the generated makefile, you can see the value of this property in the line that begins with ConfigurationCPPCompileSwitches=.

Default = Blank

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default =

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/linuxWebComponents$(LIB_EXT),
$(OMROOT)/lib/linuxWebServices$(LIB_EXT)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = Empty string

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = Empty string

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkerFlags

The property LinkerFlags allows you to define linker flags. The value of the property is inserted into the value of the property LinkSwitches. In the generated makefile, you can see the value of this property in the line that begins with LINK_FLAGS=.

Default = -lpthread -lstdc++

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -O

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ##################
##

CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ###### Predefined macros ################ RM=/bin/rm -rf MD=/bin/mkdir -p INCLUDE_QUALIFIER=-I CC=gcc -DUSE_IOSTREAM LIB_CMD=ar LINK_CMD=$(CC) LIB_FLAGS=rvu LINK_FLAGS=-lpthread -lstdc++ $OMConfigurationLinkSwitches ## ####### Context macros ################## $OMContextMacros ## ####### Predefined macros ############### $(OBJS) : $(INST_LIBS) $(OXF_LIBS) OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= $(MD) $(OBJ_DIR) CLEAN_OBJ_DIR= $(RM) $(OBJ_DIR) endif ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/linuxaomanim$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/linuxoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/linuxomcomappl$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/linuxtomtrace$(LIB_EXT) $(OMROOT)/LangCpp/lib/linuxaomtrace$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/linuxoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/linuxomcomappl$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/linuxoxf$(LIB_EXT) SOCK_LIB= else @echo An invalid Instrumentation $(INSTRUMENTATION) is specified. exit endif endif endif .SUFFIXES: $(CPP_EXT) ## ###################### Context dependencies and commands ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @$(CC) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## # Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) -o $(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS $(RM) $OMFileObjPath $(ADDITIONAL_OBJS) $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

NoneIncludeDirectories

The property NoneIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = Blank

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = Blank

NoneOxfLibs

The property NoneOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangCpp/lib/linuxoxf$(LIB_EXT)

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The

RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

TraceIncludeDirectories

The property TraceIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/aom
$(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangCpp/lib/linuxtomtrace$(LIB_EXT)
$(OMROOT)/LangCpp/lib/linuxaomtrace$(LIB_EXT)

TraceOxfLibs

The property TraceOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangCpp/lib/linuxoxfinst$(LIB_EXT)
$(OMROOT)/LangCpp/lib/linuxomcomappl$(LIB_EXT)

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMTRACER

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Checked

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Checked

Microsoft

The Microsoft metaclass contains environment properties (Compiler, framework libraries, etc.) for Microsoft Win32 compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/WIN32

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been

integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\"\etc\msmake.bat msbuild.mak build \"USE_STL=FALSE\" \"USE_PDB=FALSE\" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

COM

The COM property specifies whether the current component is a COM component. By default, this property is set to Checked for all COM components (stereotypes COM DLL, COM EXE, and COM TLB). If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

/I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default =

$(CREATE_OBJ_DIR) $(CPP) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath"

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = /Zi /Od /D "_DEBUG" /MDd /Fd"$(TARGET_NAME)"

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = /Ox /D"NDEBUG" /MD /Fd"$(TARGET_NAME)"

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files

should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default is as follows:

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT), ws2_32$(LIB_EXT)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = !INCLUDE

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from

the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default value = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet /NOLOGO

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll SimulinkLibName=$SimulinkLibName ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches=$(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF ################### Compilation flags ##################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS ################### Commands definition ######################### ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) /MACHINE:I386 ############### Generated macros ################## ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF ######################### Predefined macros ############################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX)$(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT)

$(SimulinkLibName) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POSTFIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) $(SimulinkLibName) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$(LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$(LIB_POSTFIX)$(LIB_EXT) $(SimulinkLibName) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF ################## Generated dependencies ########################

###

$OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile ########################## Linking instructions ################################

###

*$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)*

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = if exist $OMFileObjPath erase $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .obj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be

displayed alongside the message on the Build tab of the Output window.

Default = (NMAKE|LINK)(.)(fatal error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

*Default = *

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource file. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

Default = $(RC) /Fo"$(TARGET_MAIN).res" $(TARGET_MAIN)$OMRCExtension

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational

Rhapsody.

Default = .rc

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = /D "OM_REUSABLE_STATECHART_IMPLEMENTATION"

RPFrameWorkDll

The RPFrameWorkDll property determines whether the configuration uses the DLL flavor of the framework libraries. To use OXF DLLs for the creation of COM ATL components, you must set this property to Checked before you generate code. Rational Rhapsody COM ATL components use a DLL version of the OXF. This version of the OXF allows the use of multiple Rhapsody-generated DLL/executable components confined to a single process. There are three versions of the OXF DLL:

DLL Version Animation Enabled Trace Enabled oxfdll.dll No No oxfanimdll.dll Yes No oxftracedll.dll No Yes

OXF DLLs are not a part of a typical Rational Rhapsody installation, and must be built from the OXF C++ sources. To obtain these sources, you can either perform a custom installation or update the install to add the sources to your existing installation. To rebuild the framework DLLs, run the following in your $OMROOT\LangCpp folder: nmake -f msoxfanimtracedll.mak CFG=oxfdll nmake -f msoxfanimtracedll.mak CFG=oxfanimdll nmake -f msoxfanimtracedll.mak CFG=oxftracedll

To use OXF DLLS for the creation of COM ATL components, set the following component configuration properties to True before you generate code:

- CPP_CG::Microsoft::RPFrameWorkDll
- CPP_CG::MicrosoftDLL::RPFrameWorkDll this is True by default)

In addition, make sure that the following are included in the system environment path:

- OXF DLL path ($OMROOT\LangCpp\lib)
- The full path to regsrv32.exe

Without these settings, COM ATL components are not registered and cannot run. Limitations:

- Rational Rhapsody components with different instrumentation settings (both Animation and Tracing) are not supported within a single process. However, you can mix instrumented and noninstrumented DLLs, as long as you use the instrumented DLL.
- Mixing Rational Rhapsody components that link to the DLL and the library version of OXF is not supported within a single process.

Default = Cleared

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

Default = /SUBSYSTEM:console

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

MicrosoftDLL

The MicrosoftDLL metaclass contains environment properties (Compiler, framework libraries, etc.) for Microsoft Win32 compiler that creates DLLs instead of static libraries.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.

- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/WIN32

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\"\etc\msmake.bat msbuild.mak build \"USE_STL=FALSE\" \"USE_PDB=FALSE\" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

COM

The COM property specifies whether the current component is a COM component. By default, this property is set to Checked for all COM components (stereotypes COM DLL, COM EXE, and COM TLB). If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

/I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default =

$(CREATE_OBJ_DIR) $(CPP) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath"

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = /Zi /Od /D "_DEBUG" /MDd /Fd"$(TARGET_NAME)"

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = /Ox /D"NDEBUG" /MD /Fd"$(TARGET_NAME)"

DEFExtension

The DEFExtension property is a string that specifies the extension for DLL definition files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .def

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DllExtension

The DllExtension property is a string that specifies the extension for DLL files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .dll

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = !INCLUDE

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .dll

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default value = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet /NOLOGO

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ##################
##
CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet
SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll
DEF_EXT=$OMDEFExtension DLL_EXT=$OMDllExtension
ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches
$OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True"
ConfigurationCPPCompileSwitches=$(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL"
!ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF
################### Compilation flags ######################
INCLUDE_QUALIFIER=/I
LIB_PREFIX=MS ################### Commands definition ########################
RMDIR = rmdir
DLL_CMD=link.exe -dll LINK_CMD=link.exe DLL_FLAGS=$OMConfigurationLinkSwitches
LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) /MACHINE:I386 ###############
Generated macros ################ ##
$OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist
$(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE
CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF ######################## Predefined macros
##############################
!IF "$(OBJS)"
!= "" $(OBJS) : $(FLAGSFILE) $(RULESFILE) $(INST_LIBS) $(OXF_LIBS) !ENDIF LIB_EXT=.lib
LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" ==
"Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF
"$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUG /DEBUGTYPE:CV
LinkRelease=$(LinkRelease) /OPT:NOREF !ENDIF !IF "$(TIME_MODEL)" == "Simulated"
TIM_EXT= !ELSEIF "$(TIME_MODEL)" == "RealTime" TIM_EXT= !ELSE !ERROR An invalid Time
Model "$(TIME_MODEL)" is specified. !ENDIF !IF "$(INSTRUMENTATION)" == "Animation"
INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I
$(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS=
OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX)$(LIB_EXT) !ELSE
INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT)
OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$(TIM_EXT)inst$(LIB_POSTFIX)$(LIB_EXT)
$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF
SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D
"OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF
"$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=
$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POSTFIX)$(LIB_EXT) !ELSE
INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT)
$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=
$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$(TIM_EXT)inst$(LIB_POSTFIX)$(LIB_EXT)
$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF
SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS=
INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True"
OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$(LIB_POSTFIX)$(LIB_EXT) !ELSE
OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf(TIM_EXT)(LIB_POSTFIX)$(LIB_EXT)
!ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified.
!ENDIF !IF "$(COM)" == "True" COM_LIB=kernel32.lib user32.lib gdi32.lib winspool.lib comdlg32.lib
advapi32.lib shell32.lib ole32.lib oleaut32.lib uuid.lib odbc32.lib odbccp32.lib
COM_OBJS=$OMFileObjPath DEF_NAME=$(TARGET_MAIN)$(DEF_EXT)
LINK_DEF=/def:$(DEF_NAME) !ELSE COM_LIB= COM_OBJS= DEF_NAME= LINK_DEF= !ENDIF
################### Generated dependencies ########################
###
$OMContextDependencies !IF "$(TARGET_MAIN)" != "" CLEAN_MAIN_OBJ=if exist $OMFileObjPath

*erase $OMFileObjPath $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(FLAGSFILE) $(RULESFILE) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile !ELSE CLEAN_MAIN_OBJ= !ENDIF ########################## Linking instructions ############################## ## !IF "$(TARGET_NAME)" != "" $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(DLL_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $(COM_OBJS) $(DEF_NAME) $OMMakefileName @echo Building library $@ $(DLL_CMD) $(DLL_FLAGS) $(COM_LIB) $(OBJS) $(COM_OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_DEF) \ /out:$(TARGET_NAME)$(DLL_EXT) !ENDIF clean: @echo Cleanup $OMCleanOBJS $(CLEAN_MAIN_OBJ) if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)*

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = if exist $OMFileObjPath erase $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .obj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be

displayed alongside the message on the Build tab of the Output window.

Default = (NMAKE)(.)(fatal error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

*Default = *

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource file. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $(RC) /Fo"$(TARGET_MAIN).res" $(TARGET_MAIN)$OMRCExtension

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational

Rhapsody.

Default = .rc

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = /D "OM_REUSABLE_STATECHART_IMPLEMENTATION"

RPFrameWorkDll

The RPFrameWorkDll property determines whether the configuration uses the DLL flavor of the framework libraries. To use OXF DLLs for the creation of COM ATL components, you must set this property to Checked before you generate code. Rational Rhapsody COM ATL components use a DLL version of the OXF. This version of the OXF allows the use of multiple Rational Rhapsody-generated DLL/executable components confined to a single process. There are three versions of the OXF DLL:

DLL Version Animation Enabled Trace Enabled oxfdll.dll No No oxfanimdll.dll Yes No oxftracedll.dll No Yes

OXF DLLs are not a part of a typical Rational Rhapsody installation, and must be built from the OXF C++ sources. To obtain these sources, you can either perform a custom installation or update the install to add the sources to your existing installation. To rebuild the framework DLLs, run the following in your $OMROOT\LangCpp folder: nmake -f msoxfanimtracedll.mak CFG=oxfdll nmake -f msoxfanimtracedll.mak CFG=oxfanimdll nmake -f msoxfanimtracedll.mak CFG=oxftracedll

To use OXF DLLS for the creation of COM ATL components, set the following component configuration properties to True before you generate code:

- CPP_CG::Microsoft::RPFrameWorkDll
- CPP_CG::MicrosoftDLL::RPFrameWorkDll this is True by default)

In addition, make sure that the following are included in the system environment path:

- OXF DLL path ($OMROOT\LangCpp\lib)
- The full path to regsrv32.exe

Without these settings, COM ATL components are not registered and cannot run. Limitations:

- Rational Rhapsody components with different instrumentation settings (both Animation and Tracing) are not supported within a single process. However, you can mix instrumented and noninstrumented DLLs, as long as you use the instrumented DLL.
- Mixing Rational Rhapsody components that link to the DLL and the library version of OXF is not supported within a single process.

Default = Cleared

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

Default = /SUBSYSTEM:console

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

MicrosoftWinCE600

The MicrosoftWinCE600 metaclass contains environment properties (Compiler, framework libraries, etc.) for MicrosoftWinCE600 compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.

- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/WIN32

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\"\etc\mscemake.bat mscebuild.mak build $CPU \"BUILD_SET=$BuildCommandSet\" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

/I . /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX- /D _WIN32_WCE=$(CEVersion) /D "$(CEConfigName)" $(MACHINE_CPP_FLAGS) /D "_OM_NO_IOSTREAM" /D UNDER_CE=$(CEVersion) /D "UNICODE" /D "_OM_UNICODE_ONLY" $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /D "_X86_" /c

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default =

$(CPP) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath"

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = /Zi /Od /D "_DEBUG" /M$(CECrtMTDebug) /Fd"$(TARGET_NAME)"

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = /Ox /D"NDEBUG" /M$(CECrtMT) /Fd"$(TARGET_NAME

CPU

The CPU property is a string that specifies the CPU type.

Default = x86

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = cemain

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property

ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT), winsock.lib

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = !INCLUDE

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = Empty string

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msceNETmake.bat\" $makefile $maketarget x86"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default value = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet /NOLOGO

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = USE_MFC_APP_WINDOW=FALSE ############# Target type (Debug/Release) ################## ### CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease

LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(OSVERSION)" == "WCE400" CESubsystem=windowsce,4.00 CEVersion=400 CEConfigName=OPPS !ELSEIF "$(OSVERSION)" == "WCE420" CESubsystem=windowsce,4.20 CEVersion=420 CEConfigName=OPPS !ELSE !MESSAGE An invalid OSVERSION "$(OSVERSION)" is specified. !MESSAGE Please specify OSVERSION= WCE400 or WCE420 !ERROR Exiting !ENDIF CECrtMT=T CECrtMTDebug=Td CENoDefaultLib=libc.lib /nodefaultlib:libcd.lib /nodefaultlib:libcmt.lib /nodefaultlib:libcmtd.lib /nodefaultlib:msvcrt.lib /nodefaultlib:msvcrtd.lib /nodefaultlib:OldNames.lib CECorelibc=corelibc.lib !IF "$(MACHINE)" == "SH3" CPP=shcl.exe MACHINE_CPP_FLAGS=/D "SHx" /D "SH3" /D "_SH3_" MACHINE_EXT=SH !ELSEIF "$(MACHINE)" == "SH4" CPP=shcl.exe MACHINE_CPP_FLAGS=/Qsh4 /D "SHx" /D "SH4" /D "_SH4_" MACHINE_EXT=SH !ELSEIF "$(MACHINE)" == "MIPS" CPP=clmips.exe MACHINE_CPP_FLAGS=/D "MIPS" /D "_MIPS_" MACHINE_EXT=MIPS !ELSEIF "$(MACHINE)" == "ARM" CPP=clarm.exe MACHINE_CPP_FLAGS=/D "ARM" /D "_ARM_" MACHINE_EXT=PPC !ELSEIF "$(MACHINE)" == "IX86" CPP=cl.exe MACHINE_CPP_FLAGS=/D "x86" /D "_i386_" /D "_x86_" /D "i_386_" MACHINE_EXT=IX86 !ELSE !MESSAGE An invalid MACHINE "$(MACHINE)" is specified. !MESSAGE Please specify MACHINE= SH3 SH4 MIPS ARM or IX86 !ERROR Exiting !ENDIF ##################### Compilation flags ##################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=Ce$(CEVersion)$(TARGETCPU) ################### Commands definition ######################### ## LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(CECorelibc) commctrl.lib coredll.lib /SUBSYSTEM:$(CESubsystem) /MACHINE:$(MACHINE) /nodefaultlib:$(CENoDefaultLib) ############### Generated macros ################ ## $OMContextMacros ######################### Predefined macros ############################## ## $(OBJS) : $(FLAGSFILE) $(RULESFILE) $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=winsock.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=winsock.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf(TIM_EXT)(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF ################### Generated dependencies ######################### ## $OMContextDependencies $(TARGET_MAIN)$(OBJ_EXT) : $(TARGET_MAIN)$(CPP_EXT) $(OBJS) $(FLAGSFILE) $(RULESFILE) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $(TARGET_MAIN)$(CPP_EXT) !IF "$(USE_MFC_APP_WINDOW)"=="TRUE" CE_APP_FLAGS=/D USE_MFC_APP_WINDOW MAIN_ENTRY_NAME=wWinMainCRTStartup !ELSE

*MAIN_ENTRY_NAME=wWinMain CE_APP_FLAGS= !ENDIF MsCeApp$(CPP_EXT) : @echo Copying MsCeApp$(CPP_EXT) @copy $(OMROOT)\MakeTmpl\MsCeApp$(CPP_EXT) MsCeApp$(CPP_EXT) MsCeApp$(OBJ_EXT) : MsCeApp$(CPP_EXT) $(CPP) $(CE_APP_FLAGS) $(ConfigurationCPPCompileSwitches) MsCeApp$(CPP_EXT) ########################## Linking instructions ################################ ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $(TARGET_MAIN)$(OBJ_EXT) MsCeApp$(OBJ_EXT) $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $(TARGET_MAIN)$(OBJ_EXT) MsCeApp$(OBJ_EXT) /entry:"$(MAIN_ENTRY_NAME)" /base:"0x00010000" $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT)*

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = if exist $OMFileObjPath erase $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated

makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .obj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource

file. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $(RC) /Fo"$(TARGET_MAIN).res" $(TARGET_MAIN)$OMRCExtension

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .rc

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = /D "OM_REUSABLE_STATECHART_IMPLEMENTATION"

RPFrameWorkDll

The RPFrameWorkDll property determines whether the configuration uses the DLL flavor of the framework libraries. To use OXF DLLs for the creation of COM ATL components, you must set this property to Checked before you generate code. Rational Rhapsody COM ATL components use a DLL version of the OXF. This version of the OXF allows the use of multiple Rational Rhapsody-generated DLL/executable components confined to a single process. There are three versions of the OXF DLL:

DLL Version Animation Enabled Trace Enabled oxfdll.dll No No oxfanimdll.dll Yes No oxftracedll.dll No Yes

OXF DLLs are not a part of a typical Rational Rhapsody installation, and must be built from the OXF C++ sources. To obtain these sources, you can either perform a custom installation or update the install to

add the sources to your existing installation. To rebuild the framework DLLs, run the following in your $OMROOT\LangCpp folder: nmake -f msoxfanimtracedll.mak CFG=oxfdll nmake -f msoxfanimtracedll.mak CFG=oxfanimdll nmake -f msoxfanimtracedll.mak CFG=oxftracedll

To use OXF DLLS for the creation of COM ATL components, set the following component configuration properties to True before you generate code:

- CPP_CG::Microsoft::RPFrameWorkDll
- CPP_CG::MicrosoftDLL::RPFrameWorkDll this is True by default)

In addition, make sure that the following are included in the system environment path:

- OXF DLL path ($OMROOT\LangCpp\lib)
- The full path to regsrv32.exe

Without these settings, COM ATL components are not registered and cannot run. Limitations:

- Rational Rhapsody components with different instrumentation settings (both Animation and Tracing) are not supported within a single process. However, you can mix instrumented and noninstrumented DLLs, as long as you use the instrumented DLL.
- Mixing Rational Rhapsody components that link to the DLL and the library version of OXF is not supported within a single process.

Default = Cleared

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

Default = /SUBSYSTEM:console

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build

settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

MSStandardLibrary

The MSStandardLibrary metaclass contains environment properties (Compiler, framework libraries, etc.) for Microsoft Win32 compiler and uses the standard iostreams (relevant only for VC++ 6.0 users).

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the

path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/WIN32

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 thread dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release

version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"\"$OMROOT\"\etc\msmake.bat msbuild.mak build \"USE_STL=TRUE\" \"USE_PDB=FALSE\"\""

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

/I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" /D "OM_USE_STL" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default =

$(CREATE_OBJ_DIR) $(CPP) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath"

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = /Zi /Od /D "_DEBUG" /MDd /Fd"$(TARGET_NAME)"

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = /Ox /D"NDEBUG" /MD /Fd"$(TARGET_NAME)"

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default = $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\MSWebServices$(LIB_POSTFIX)$(LIB_EXT), ws2_32$(LIB_EXT)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = !INCLUDE

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default value = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet /NOLOGO

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

*Default = ############ Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ################## Compilation flags ####################### ## LIB_PREFIX=MSStl INCLUDE_QUALIFIER=/I ################## Commands definition ######################### ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches /SUBSYSTEM:console /MACHINE:I386 /nodefaultlib:"libc.lib" ############### Generated macros ################## ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF ######################### Predefined macros ############################## ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF ################### Generated dependencies ######################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile ############################ Linking instructions ############################### ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)*

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = if exist $OMFileObjPath erase $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .obj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (NMAKE)(.)(fatal error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

*Default = *

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = /D "OM_REUSABLE_STATECHART_IMPLEMENTATION"

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

Multi4Linux

Environment settings (Compiler, framework libraries, etc.) for Multi4Linux compiler.

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches. The default values are as follows:

Environment Default Value INTEGRITY :optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 IntegrityESTL :optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 :cx_mode=extended_embedded :cx_lib=eece :stdcxxincdirs=$(INTEGRITY_ROOT)\eecxx :stdcxxincdirs=$(INTEGRITY_ROOT)\ansi MultiWin32 :cx_template_option=noimplicit :add_output_ext=checked :cx_e_option=msgnumbers :cx_option=exceptions :check=bounds :check=assignbound :check=nilderef :cx_template=local :cx_remark=14 :cx_remark=161 :cx_remark=837 :cx_remark=817 :cx_remark=815 :cx_remark=47 :cx_remark=69 :cx_remark=830 :cx_remark=550 :prelink.args=-r :prelink.args=-X7

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model. The default value for Ada is as follows:

:target_os=integrity :ada_library=full :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model. The default values are as follows:

Environment Default Value INTEGRITY :target_os=integrity IntegrityESTL MultiWin32 Empty MultiLine

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link. The default value for ADA INTEGRITY metaclass is "sim800."

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

The default value for MultiWin32 is DebugNoExp; for the other environments, the default value is Debug.

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property. The default values are as follows:

Environment Default Compile Switches Borland -I$OMDefaultSpecificationDirectory -I$(BCROOT)\INCLUDE;.;"$(OMROOT)\LangCpp"; "$(OMROOT)\LangCpp\oxf";"$(OMROOT)\LangCpp\omCom"; -D_RTLDLL;_AFXDLL;WIN32;_CONSOLE;_MBCS; WINDOWS;BORLAND;_BOOLEAN $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) $OMCPPCompileCommandSet -c Linux MontaVista -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c Microsoft MicrosoftDLL /I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MicrosoftWinCE /I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX- /D _WIN32_WCE=$(CEVersion) /D "$(CEConfigName)" $(MACHINE_CPP_FLAGS) /D "_OM_NO_IOSTREAM" /D UNDER_CE=$(CEVersion) /D "UNICODE" /D "_OM_UNICODE_ONLY" $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MicrosoftWinCE600 /I . /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX- /D _WIN32_WCE=$(CEVersion) /D "$(CEConfigName)" $(MACHINE_CPP_FLAGS) /D "_OM_NO_IOSTREAM" /D UNDER_CE=$(CEVersion) /D "UNICODE" /D "_OM_UNICODE_ONLY" $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /D "_X86_" /c MSStandardLibrary /I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" /D "OM_USE_STL" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MultiWin32 ${CPPCompileDebugNoExp} $CPPAdditionalCompileSwitches Nucleus PLUS-PPC -v -c -DPLUS -DUSE_IOSTREAM -D__DIAB -t$(CPU) -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)\LangCpp -I$(OMROOT)\LangCpp\oxf -I$(ATI_DIR) -Xmismatch-warning -Xno-common $OMCPPCompileCommandSet $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) OsePPCDiab OseSfk -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)$/LangCpp $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) PsosPPC PsosX86 $OMCPPCompileCommandSet QNXNeutrinoCW -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c QNXNeutrinoGCC Solaris2 Solaris2GNU -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)

-I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c VxWorks -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default values are as follows: Environment Default Value Borland PsosX86 ToTalNumberOfTokens=5,FileTokenPosition=4,LineTokenPosition=5 GNAT ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2 JDK Linux MontaVista QNXNeutrinoCW QNXNeutrinoGCC Solaris2GNU SPARK VxWorks Microsoft ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2 MicrosoftDLL MicrosoftWinCE600 MSStandardLibrary NucleusPLUS-PPC OsePPCDiab OseSfk PsosPPC Solaris2 INTEGRITY (Ada) ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3 MultiWin32 (Ada) OBJECTADA RAVEN_PPC

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The default value for QNXNeutrinoCW is False; for the other environments, the default value is True.

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The default values are as follows:

Environment Default Value QNXNeutrinoCW $OMROOT/DLLs/CodeWarriorIDE.dll INTEGRITY Empty string IntegrityESTL VxWorks $OMROOT/DLLs/TornadoIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(C++ Default = .cpp; for OsePPCDiab and OseSfk, the default is .cc)

IntegrityLinkFile

The name of the Integrity link file that should be added to the makefile template.

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file. The default values are as follows:

Environment Default Value Borland "$executable" GNAT Microsoft MicrosoftDLL MSStandardLibrary MultiWin32 (Ada) NucleusPLUS-PPC OBJECTADA RAVEN_PPC SPARK INTEGRITY Empty string IntegrityESTL MicrosoftWinCE600 MontaVista JDK "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\jdkrun.bat\" $makefile Main$ComponentName" Linux $executable MultiWin32 (C++) QNXNeutrinoCW QNXNeutrinoGCC MicrosoftWinCE "$OMROOT\etc\msceRun.bat" $executable IX86EM OsePPCDiab "$OMROOT/etc/osesfkRun.bat" $executable OseSfk Solaris2 xterm -e $executable Solaris2GNU

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

The InvokeMake default values are as follows:

Environment Default Value Borland $OMROOT/etc/Executer.exe "\"$OMROOT\etc\bc5make.bat\" $makefile $maketarget" GNAT "$makefile" $maketarget INTEGRITY ESTL "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" $makefile $maketarget" Integrity JDK "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\jdkmake.bat\" $makefile $maketarget" Linux $OMROOT/etc/linuxmake $makefile $maketarget Microsoft "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget" MicrosoftDLL MSStandardLibrary MicrosoftWinCE "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\mscemake.bat\" $makefile $maketarget IX86EM" MicrosoftWinCE600 "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msceNETmake.bat\" $makefile $maketarget x86" MontaVista "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\mvlinuxmake.bat\" $makefile $maketarget" MultiWin32 (Ada) "$OMROOT/etc/Executer.exe" "$OMROOT\etc\AdaMultiWin32Make.bat $makefile $maketarget" OBJECTADA "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaMake.bat $makefile $maketarget" OsePPCDiab "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\oseppcdiabmake.bat\" $makefile $maketarget" OseSfk "$OMROOT/etc/Executer.exe"

"\"$OMROOT\etc\osesfkmake.bat\" $makefile $maketarget" PsosPPC "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\psppcmake.bat\" $makefile $maketarget" PsosX86 "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\psx86make.bat\" $makefile $maketarget" QNXNeutrinoCW "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\qnxcwmake.bat\" $makefile $maketarget" QNXNeutrinoGCC Empty string RAVEN_PPC "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaRavenPPCMake.bat $makefile $maketarget" Solaris2 $OMROOT/etc/sol2make $makefile $maketarget Solaris2GNU SPARK "$OMROOT/etc/Executer.exe" "$OMROOT\etc\SPARKMake.bat $makefile $maketarget" VxWorks "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vxmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment. The default values are as follows:

Environment Default Value Borland .lib Microsoft MicrosoftWinCE600 MSStandardLibrary MultiWin32 NucleusPLUS-PPC OBJECTADA OseSfk PsosX86 GNAT .a INTEGRITY IntegrityESTL Linux MontaVista OsePPCDiab PsosPPC QNXNeutrinoCW QNXNeutrinoGCC RAVEN_PPC Solaris2 Solaris2GNU VxWorks JDK Empty string SPARK MicrosoftDLL .dll

LibPrefix

Combines all the prefixes of library names.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

Environment Default Value Borland $OMLinkCommandSet Linux MontaVista MultiWin32 (C++) NucleusPLUS-PPC OsePPCDiab PsosPPC PsosX86 QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks INTEGRITY --one_instantiation_per_object $OMLinkCommandSet -cpu=$(TARGET_CPU) -map IntegrityESTL Microsoft $OMLinkCommandSet /NOLOGO MicrosoftDLL MicrosoftWinCE600 MSStandardLibrary OseSfk -nologo $OMLinkCommandSet QNXNeutrinoCW -static

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .gpj)

MakeFileContentForExe1

The content of the makefiles, in case of executable component type.

MakeFileContentForExe2

The content of the makefiles, in case of executable component type.

MakeFileContentForLib1

The content of the makefiles, in case of library component type.

MakeFileContentForLib2

The content of the makefiles, in case of library component type.

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

The name of the makefiles, in case of executable component type.

MakeFileNameForExe2

The name of the makefiles, in case of executable component type.

MakeFileNameForLib1

The name of the makefiles, in case of library component type.

MakeFileNameForLib2

The name of the makefiles, in case of library component type.

MultipleAddressSpacesIntFileContent

The content of the MultipleAddressSpacesIntFileName file.

MultipleAddressSpacesIntFileName

a file with this name is created in case of multiple address space compilation.

MultipleAddressSpacesLibraries

names of libraries to add in case of multiple address space usage.

MultipleAddressSpacesPrefix

A prefix that is added to libraries in case of multiple address space compilation. OMMultipleAddressSpacesPrefix keyword will add this prefix when needed.

MultipleAddressSpacesSwitches

A switch for multiple address space compilation.The makefile template can add it directly but it is preferred to use the keyword OMMultipleAddressSpacesSwitches – that checks whether this switch should be added.

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive. The default values are as follows:

Environment Default Value Borland Cleared GNAT INTEGRITY IntegrityESTL JDK Microsoft MicrosoftDLL MicrosoftWinCE600 MSStandardLibrary MultiWin32 NucleusPLUS-PPC OseSfk OsePPCDiab PsosPPC PsosX86 RAVEN_PPC SPARK VxWorks Linux Checked MontaVista QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. The default values are as follows:

Environment Default Value Borland PsosX86 ToTalNumberOfTokens=5,FileTokenPosition=4,LineTokenPosition=5 GNAT ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2 JDK Linux MontaVista QNXNeutrinoCW QNXNeutrinoGCC Solaris2GNU SPARK VxWorks IntegrityESTL ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2 Microsoft MicrosoftDLL MicrosoftWinCE600 MSStandardLibrary NucleusPLUS-PPC OsePPCDiab OseSfk PsosPPC Solaris2 INTEGRITY (Ada) ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=3 MultiWin32 (Ada) OBJECTADA RAVEN_PPC

PosixSharedMemoryFiles

This list is copied only in case of multiple address space compilation.

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment. The extension ".ads" is the default for Ada.

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

For INTEGRITY, OBJECTADA, RAVEN_PPC, and SPARK the default value is Cleared; for the other environments, the default value is Checked.

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

The default value for the following environments is Cleared:

Borland GNAT INTEGRITY IntegrityESTL Microsoft MicrosoftDLL MSStandardLibrary MultiWin32

The default value for the following environments is Checked:

Linux MicrosoftWinCE600 MontaVista NucleusPLUS-PPC OsePPCDiab OseSfk PsosPPC PsosX86 QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks

WebInstLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on.

Multi4Win32

The Multi4Win32 metaclass contains environment properties (Compiler, framework libraries, etc.) for GHS MULTI 4.0.X Win32 compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the

framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMRoot)\LangCpp\osconfig\MultiWin32

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomAnim$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -D_OMINSTRUMENT

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags.

Default = Empty string

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

-threading=multiple -I. --exceptions --no_implicit_include --display_error_number --diag_remark 14,161,837,817,815,47,69,830,550 -tlocal -prelink.args=-r -prelink.args=-X7

BLDIncludeAdditionalBLD

The IncludeAdditional property enables you to specify additional build options.

Default = Empty MultiLine

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = Empty String

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that are used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).

- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = DebugNoExp

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\MultiWin32Make.bat\" MultiWin32Build.bat buildLibs "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

COM

The COM property specifies whether the current component is a COM component. By default, this property is set to Checked for all COM components (stereotypes COM DLL, COM EXE, and COM TLB). If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows.

Default = Cleared

CompileDebug

The CompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = Empty string

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the

CPPCompileSwitches property.

Default =

-DUSE_IOSTREAM -DHAS_NO_EXP

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by the software. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -D_DEBUG -G

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty MultiLine

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the software animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

EnvironmentVarName

The EnvironmentVarName property specifies the name of the global variable that you must define in order to use the Embedded C++ compiler. It is used by the MultiMakefileGenerator. The value replaces the $EnvironmentVarName value> keyword inside the property value BLDAdditionalOptions.

Default = MULTI_ROOT

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

Default = Empty string

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

Default =Multi4Win32

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(LibPrefix)WebComponents$(BLDTarget)$OMLibSuffix$LibExtension , $OMRoot/lib/$(FrameworkLibPrefix)WebServices$(BLDTarget)$OMLibSuffix$LibExtension

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = !INCLUDE

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Multi4Win32Make.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored.

Default = $OMROOT/etc/MultiMakefileGenerator.exe

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.

- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LibPrefix

Combines all the prefixes of library names.

Default = $(FrameworkLibPrefix)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string (blank)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default value = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = Empty string

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .gpj)

MakeFileContentForExe1

The content of the makefiles, in case of executable component type.

Default =

#!gbuild primaryTarget=$PrimaryTarget [Project] -object_dir=$ObjectsDirectory $MakeFileNameForExe2

MakeFileContentForExe2

The content of the makefiles, in case of executable component type.

Default =

#!gbuild [Program] -o $OMTargetName$ExeExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions $BLDIncludeAdditionalBLD -I$(OMRoot)/LangCpp -L$(OMRoot)/LangCpp/lib $OMUserIncludePath $LinkSwitches $OMCompilationFlag $CompileSwitches $OMReusableFlag $OMInstrumentationFlags $OMInstrumentationLibs $BLDAdditionalDefines $OMUserLibs $OMMainFiles$ImpExtension $OMSrcFiles

MakeFileContentForLib1

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -object_dir=$ObjectsDirectory $MakeFileNameForLib2

MakeFileContentForLib2

The content of the makefiles, in case of library component type.

The default is as follows:

#!gbuild [Library] -o $OMTargetName$LibExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions $BLDIncludeAdditionalBLD -I$(OMRoot)/LangCpp $OMUserIncludePath $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $OMReusableFlag $BLDAdditionalDefines $OMSrcFiles

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile

generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForExe2

The name of the makefiles, in case of executable component type.

Default = $(OMTargetName)_program$MakeExtension

MakeFileNameForLib1

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)$MakeExtension

MakeFileNameForLib2

The name of the makefiles, in case of library component type.

Default = $(OMTargetName)_library$MakeExtension

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

Default = -lwsock32.lib

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is

set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = Empty string

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = work

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .obj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (error|warning) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^"]+)", line ([0-9]+)

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (Error)[:] (build failed)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that

should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource file. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .rc

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

Default = R

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the

machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = OM_REUSABLE_STATECHART_IMPLEMENTATIO

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

Default = /SUBSYSTEM:console

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)TomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -DOMTRACER

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors check box (located in the configuration Initialization tab).

Default = Cleared

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

MultiWin32

The MultiWin32 metaclass contains the environment settings (Compiler, framework libraries, etc.) for GHS MULTI 3.5 Win32 compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)\LangCpp\osconfig\MultiWin32

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags.

Default = Empty string

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default =

:cx_template_option=noimplicit :add_output_ext=true :cx_e_option=msgnumbers :cx_option=exceptions :check=bounds :check=assignbound :check=nilderef :cx_template=local :cx_remark=14 :cx_remark=161 :cx_remark=837 :cx_remark=817 :cx_remark=815 :cx_remark=47 :cx_remark=69 :cx_remark=830 :cx_remark=550 :prelink.args=-r :prelink.args=-X7 :defines=OM_STL

BLDIncludeAdditionalBLD

The IncludeAdditional property enables you to specify additional build options.

Default = Empty MultiLine

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model. The default values for the C++ INTEGRITY metaclass are as follows:

:defines=_DEBUG :target_os=integrity :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

Default =

:defines=_DEBUG :target_os=integrity :integrity_option=dynamic :staticlink=true

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

Default = Empty string

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to

True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = DebugNoExp

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\MultiWin32Make.bat\" MultiWin32Build.bat buildLibs bld "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

COM

The COM property specifies whether the current component is a COM component. By default, this property is set to Checked for all COM components (stereotypes COM DLL, COM EXE, and COM TLB).

If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows.

Default = Cleared

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the software animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression

- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

Default =MultiWin32

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = $executable

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\MultiWin32Make.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored.

Default = $OMROOT/etc/MultiMakefileGenerator

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .bld)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = Empty MultiLine

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = Empty string

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = obj_dir

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .obj

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (error|warning) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. The default values are as follows:

Default = ([^"]+)", line ([0-9]+)

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (Error)[:] (build failed)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource file. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = .rc

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = OM_REUSABLE_STATECHART_IMPLEMENTATION.

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

Default = /SUBSYSTEM:console

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

NucleusPLUS-PPC

The NucleusPLUS-PPC metaclass contains environment settings (Compiler, framework libraries, etc.) for NucleusPLUS RTOS compiled for PPC, using Diab compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/Nucleus

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to

True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\numake.bat" nubuild.mak buildLibs \"CPU=$CPU\" \"BUILD_SET=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then the software calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-v -c -DPLUS -DUSE_IOSTREAM -D__DIAB -t$(CPU) -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)\LangCpp -I$(OMROOT)\LangCpp\oxf -I$(ATI_DIR) -Xmismatch-warning -Xno-common $OMCPPCompileCommandSet $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES)

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Default =

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

$(CPP) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g -D_DEBUG -DASSERT_DEBUG

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -DNDEBUG

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $(CREATE_OBJ_DIR) $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the software animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = numain

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .elf)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default is as follows:

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = .INCLUDE:

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$executable"

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\numake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .lib

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Custom User Settings ########################## ## .IMPORT .IGNORE : ATI_DIR CPU=PPC860ES LIBS=$(ATI_DIR)\PLUS\O\PLUS.LIB -ld DLDFILE=$(OMROOT)\MakeTmpl\nuos.dld NU_SOCK_LIB=$(ATI_DIR)\lib\net.lib $(ATI_DIR)\lib\pquicc.lib ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ################## Compilation flags ###################### ## INCLUDE_QUALIFIER=-I LIB_PREFIX=Nu ################## Commands definition ########################## ## CPP=dcc.exe LIB_CMD=dar.exe LINK_CMD=dld.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches CP=cp RM=rm ############### Generated macros ################# ## $OMContextMacros ######################### Predefined macros ############################ ## OBJ_DIR=$OMObjectsDir .IF "$(OBJ_DIR)"!="" create_obj_dir: @[@echo off @if not exist $(OBJ_DIR) mkdir $(OBJ_DIR)] CREATE_OBJ_DIR= create_obj_dir CLEAN_OBJ_DIR=if exist $(OBJ_DIR) rmdir $(OBJ_DIR) .ELIF CREATE_OBJ_DIR= CLEAN_OBJ_DIR= .ENDIF $(OBJS) : $(FLAGSFILE) $(RULESFILE) $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= .IF "$(BuildSet)"=="Release" LIB_POSTFIX=R .ENDIF .IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=-DOMANIMATOR INST_INCLUDES=-I$(OMROOT)\LangCpp\aom -I$(OMROOT)\LangCpp\tom INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=$(NU_SOCK_LIB) .ELIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=-DOMTRACER INST_INCLUDES=-I$(OMROOT)\LangCpp\aom -I$(OMROOT)\LangCpp\tom INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinstt$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=$(NU_SOCK_LIB) .ELIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB=$(NU_SOCK_LIB) .ELSE emsg2: @[@echo An invalid Instrumentation $(INSTRUMENTATION) is specified. @error] .ENDIF NU_ADAPTOR_OBJ=NuAppInit$(OBJ_EXT) $(NU_ADAPTOR_OBJ) : NuAppInit$(CPP_EXT) $(CPP) $(ConfigurationCPPCompileSwitches) NuAppInit$(CPP_EXT) NuAppInit$(CPP_EXT) : $(OMROOT)/MakeTmpl/NuAppInit$(CPP_EXT) $(CP) "$<" $@ ################## Generated dependencies ######################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(NU_ADAPTOR_OBJ) $(FLAGSFILE) $(RULESFILE) $(CPP) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ########################## Linking instructions ################################ ##

LNK_OPTIONS_FILE=linker.opt $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $(NU_ADAPTOR_OBJ) $OMFileObjPath $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) echo $(LINK_FLAGS) -t$(CPU):simple -o $(TARGET_NAME)$(EXE_EXT) > $(LNK_OPTIONS_FILE) for %F in (.o) do @echo %F >> $(LNK_OPTIONS_FILE) echo $(ADDITIONAL_OBJS) $(LIBS) $(INST_LIBS) $(OXF_LIBS) $(SOCK_LIB) >> $(LNK_OPTIONS_FILE) $(LINK_CMD) -@$(LNK_OPTIONS_FILE) $(ATI_DIR)\PLUS\O\PLUS.LIB -ld -lc -lios -lram $(DLDFILE) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) -r $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) $(LIB_CMD) -sR $(TARGET_NAME)$(LIB_EXT) clean: @[@echo off @echo Cleanup $OMCleanOBJS @if exist $OMFileObjPath $(RM) $OMFileObjPath @if exist $(LNK_OPTIONS_FILE) $(RM) $(LNK_OPTIONS_FILE) @if exist NuAppInit$(OBJ_EXT) $(RM) NuAppInit$(OBJ_EXT) @if exist $(OBJ_DIR)/*$(OBJ_EXT) $(RM) $(OBJ_DIR)/*$(OBJ_EXT) @if exist $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(LIB_EXT) @if exist $(TARGET_NAME)$(EXE_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)]*

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = @if exist $OMFileObjPath $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (warning)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

Operation

The Operation metaclass contains properties that control operations.

ActivityReferenceToAttributes

The ActivityReferenceToAttributes property specifies whether Rational Rhapsody should generate references in the functor object, thereby giving you direct access to the attributes of the class that owns the

modeled operation (without the need for this_). See the section on activity diagrams in the Rational Rhapsody Help for detailed information about modeled operations and functor classes.

Default = Checked

AnimAllowInvocation

The AnimAllowInvocation property specifies whether primitive and triggered operations can be called during instrumentation. If an operation is called during animation, its return value is displayed in the output window; if it is traced, the return value is displayed in the console. The possible values are as follows:

- All - Enable all operation calls, regardless of visibility.
- None - Do not enable operation calls.
- Public - Enable calls to public operations only.
- Protected - Enable calls to protected operations only.

Default = None

AnimateTriggeredOperationReturnValue

The property AnimateTriggeredOperationReturnValue allows you to specify that the return values of triggered operations should be displayed in animated sequence diagrams.

Default = Checked

DeclarationModifier

The property DeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in operation declarations. Keywords that appear between the return type and the operation name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and PostDeclarationModifier.

Default = Blank

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

EntryCondition

The EntryCondition property specifies the task guard. Default = Empty string

GenerateImplementation

The GenerateImplementation property specifies whether to generate the body for the operation. To generate Import pragmas in Rational Rhapsody Developer for Ada, set this property to False and add the "pragma..." declaration in the CPP_CG::Operation::SpecificationEpilog property. (Default = True)

ImplementActivityDiagram

The ImplementActivity Diagram property enables or disables code generation for activity diagrams.

Default = Cleared

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationName

The ImplementationName property enables you to give an operation one model name and generate it with another name. It is introduced as a workaround that enables you to generate const and non-const operations with the same name. For example:

- Create a class A.
- Add a non-const operation f().
- Add a const operation f_const().
- Set the CPP_CG::Operation::ImplementationName property for f_const() to “f.”
- Generate the code.

The resulting code is as follows: class A { ... void f(); /* the non const f */ ... void f() const; /* actually f_const() */ ... }; The creation of two operations with the same signature, differing only in whether it is a const, is a common practice in C++, especially for STL users.

Default = Empty string

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

ImplementFlowchart

ImplementFlowchart is a boolean property that specifies whether or not code should be generated for the flow charts created by the user. It can be set at the individual operation level or at higher levels, such as class or package.

Default = Checked

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for C++ The possible values for the Inline property are as follows:

- none - The operation is not generated inline.
- in_header - The operation is generated inline in the specification file.
- in_source - The operation is generated inline in the implementation file.
- in_declaration - A class operation is generated inline in the class declaration. A global function is generated inline in the package specification file.

Inlining an operation in the header might cause problems if the function body refers to other classes. For example, if the inlined code refers to another class (via a pointer such as itsRelatedClass), inlined code generated in a header might not compile. The implementation file for the class would have an #include for RelatedClass, but the specification file would not. The workaround is to create a Usage dependency of the class with the inlined function on the related class. This forces an #include of the related class to be generated in the header of the dependent class with the inlined function.

Default = none

IsAnimationHelper

The IsAnimationHelper property indicates whether the operation should be generated only when animating the model. (Default = False)

IsEntry

The IsEntry property indicates whether the operation is a task entry or a regular operation in AdaTask and AdaTaskType classes. (Default = False)

IsExplicit

The boolean property IsExplicit allows you to specify that a constructor is an explicit constructor.

Default = Cleared

IsNative

The IsNative property specifies whether the Java modifier "native" should be added to an operation in the source file. The body of such operations, if specified, is ignored by the code generator. (Default = False)

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation.

Default = Empty string

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations

property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = None

Me

The Me property specifies the name of the first argument to operations generated in C. (Default = me)

MeDeclType

The MeDeclType property is a string that specifies the type of the first argument to operations generated in C, as a pointer to an object or object type. The default value is as follows: $objectName* const The variable $objectName is replaced with the name of the object or object type.

PostDeclarationModifier

The property PostDeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in operation declarations. Keywords that appear after the operation argument list are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and DeclarationModifier.

Default = Blank

PreDeclarationModifier

The property PreDeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in operation declarations. Keywords that appear before the return type are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties DeclarationModifier and PostDeclarationModifier.

Default = Blank

PrivateQualifier

The PrivateQualifier property specifies the qualifier that is printed at the beginning of a private operation declaration or definition. You can set this property to an empty string to prevent the generation of the static qualifier in the private function declaration or definition.

Default = static

ProtectedName

The ProtectedName property specifies the pattern used to generate names of private operations in C. The default value is as follows: $opName The $opName variable specifies the name of the operation. For example, the generated name of a private operation go() of an object A is generated as: go()

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C. The default value is as follows: $objectName_$opName The $objectName variable specifies the name of the object; the $opName variable specifies the name of the operation. For example, the generated name of a public operation go() of an object A is generated as: A_go()

PublicQualifier

The PublicQualifier property specifies the qualifier that is printed at the beginning of a public operation declaration or definition. Note that the Static check box in the operation dialog UI is disabled in Rational Rhapsody Developer for C because the check box is associated with class-wide semantics that are not supported by Rational Rhapsody Developer for C. When loading models from previous versions, the Static check box is cleared; if the operation is public, the C_CG::Operation::PublicQualifier property value is set to Static in order to generate the same code.

Default = Empty string

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

RenamesKind

The RenamesKind property specifies whether the renaming of the operation designated in the CPP_CG::Operation::Renames property is "as specification" or "as body."

Default = Specification

ReturnTypeByAccess

The ReturnTypeByAccess property determines whether the return type is generated as an access type or a regular type. Note that this property is applicable only to classes for which an access type is generated. (Default = False)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif

- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

TaskDefaultScheme

The TaskDefaultScheme property sets the task default entry scheme. The possible values are as follows:

- Conditional
- Timed
- None

Default = None

TaskDefaultSchemeDelayStatement

The TaskDefaultScheme property sets the task default entry statement for timed entry schemes.

Default = Empty MultiLine

ThisByAccess

The ThisByAccess property specifies whether to pass the this parameter as an access mode parameter for a non-static operation. (Default = False)

ThisName

The ThisName property enables you to specify the name of the this parameter, which specifies the instance. (Default = this)

ThrowExceptions

The ThrowExceptions property specifies the exceptions that an operation can throw. Separate multiple exceptions with commas.

Default = Empty string

VirtualMethodGenerationScheme

The VirtualMethodGenerationScheme property enables backward-compatibility mode for methods of interface and abstract classes. The possible values are as follows:

- Default - The class type is class-wide, but the this parameters are not.
- ClassWideOperations - The class type is not class-wide, but the this parameters are.

Default = Default

OsePPCDiab

The OsePPCDiab metaclass contains environment settings (Compiler, framework libraries, etc.) for OSE Delta RTOS compiled for PPC, using Diab compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/OSE

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)$/LangCpp $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES)

ConvertHostToIP

The ConvertHostToIP property specifies whether to convert the host name to an IP number. This is necessary because pSOSystem does not include a name service.

Default = Checked

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty MultiLine

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default =

$OMFileObjPath : $OMFileImpPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = rhposemain

See also the definition of the EntryPointDeclarationModifier property for more information.

EntryPointDeclarationModifier

The EntryPointDeclarationModifier property specifies a modifier for the entry point declaration. This property allows generation of the main() function in the specified syntax.

To modify the main() signature implemented in the OSE adapter, do the following:

- Add the property EntryPointDeclarationModifier to your environment properties and set it to the main return value and name. For example: "int main"
- Set the EntryPoint property to the main arguments. For example: "int a, long b, char**"
- Generate the code.

You will get the following main() declaration:

int main(int a, long b, char** c) { ... }

Default = OS_PROCESS

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location

of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .elf)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = Empty string

GeneratedAllDependencyRule

The GeneratedAllDependencyRule property specifies whether to automatically generate the “all:” rule as part of the expansion of the $OMContextMacros keyword in the makefile. If this is Cleared, you can define the makefile macros manually.

Default = Cleared

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values is as follows:

Default = .cc

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$OMROOT/etc/osesfkRun.bat" $executable

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\oseppcdiabmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.

- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default value = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MainIncludes

The MainIncludes property is a string that specifies the files that need to be included in the main program generated for an application.

Default = <ose.h>

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mk)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ###### Predefined macros ################ INCLUDE_QUALIFIER=-I LIB_CMD=dar LIB_FLAGS=rv LINK_FLAGS=$OMConfigurationLinkSwitches ### ####### Context macros ################## $OMContextMacros ## #.PHONY : all .DEFAULT : all LIB_PREFIX = OSE LIB_POSTFIX = PPC$(PROCESSOR) .IF $(TARGET_TYPE) == Executable OBJS += $OMFileObjPath .END .IF $(INSTRUMENTATION) == Animation INST_FLAGS=-DOMANIMATOR INST_INCLUDES=$(INCLUDE_QUALIFIER)$(OMROOT)$/LangCpp$/aom $(INCLUDE_QUALIFIER)$(OMROOT)$/LangCpp$/tom INST_LIBS= $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)omcomappl$(LIB_POSTFIX)$(LIB_EXT) .ELIF $(INSTRUMENTATION) == Tracing INST_FLAGS=-DOMTRACER INST_INCLUDES=$(INCLUDE_QUALIFIER)$(OMROOT)$/LangCpp$/aom $(INCLUDE_QUALIFIER)$(OMROOT)$/LangCpp$/tom INST_LIBS=$(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) .ELIF $(INSTRUMENTATION) == None INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)oxf$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB= .ELSE MAKEFILE_ERROR = yes ERROR_TYPE = user ERROR_MSG = An invalid Instrumentation INSTRUMENTATION=$(INSTRUMENTATION) is specified. .END ## usage

*.PHONY: $(ECHO)Available make targets are: $(ECHOEND) $(ECHO) clean - delete the directory $(OBJ) and all its files.$(ECHOEND) $(ECHO) all - build executable file.$(ECHOEND) $(ECHOEMPTY) ## # SETS HOST TO EITHER UNIX OR WIN32 ## HOST = $(eq,$(OS),unix UNIX WIN32) ## # READ THE USER CONFIGURATION FILE ## # The USERCONF macro can be overrided on the command line. E.g. # > dmake USERCONF=~/myconf.mk all #USERCONF *= .$/userconf.mk #include $(USERCONF) ## # THE USER CONFIGURATION ## USERCONF *= $(OMROOT)$/MakeTmpl$/oseDiabPPCconf.mk include $(USERCONF) ## CXXFLAGS += $(ConfigurationCPPCompileSwitches) .IF $(COMPILER) == DIAB DEFINES += -D__DIAB .END LIBRARIES += $(INST_LIBS) $(OXF_LIBS) $(LIBS) $(SOCK_LIB) ## OBJ = .$/obj OBJ_SUBDIR = .IF $(OBJ) != $(NULL) .IF $(OBJ) != . $(OBJ) .IGNORE: $(ECHO)Create: $@ $(ECHOEND) $(MKDIR) $(OBJ) .IF $(OBJ_SUBDIR) != $(NULL) $(MKDIR) $@ .END all: $(OBJ) CLEAN_OBJ .PHONY: $(RMDIR) $(OBJ) CLEAN += CLEAN_OBJ .END .END SRC = . INCLUDE += -I$(OBJ) INCLUDE += -I. EXAMPLES_COMMON_CONF *= $(EXAMPLES_COMMON)$/conf EXAMPLES_COMMON_INCLUDE *= $(EXAMPLES_COMMON)$/include EXAMPLES_COMMON_MAKE *= $(EXAMPLES_COMMON)$/make EXAMPLES_COMMON_SRC *= $(EXAMPLES_COMMON)$/src INCLUDE += -I$(EXAMPLES_COMMON_INCLUDE) # Inclusion of your common settings. # In this file, you can enter constants to be used for all # examples, e.g. COMPILER, COMPILERROOT etc. include $(EXAMPLES_COMMON_MAKE)$/common_settings.mk .IF $(HOST) == UNIX include $(EXAMPLES_COMMON_MAKE)$/tools-unix.mk .ELSE include $(EXAMPLES_COMMON_MAKE)$/tools-win32.mk .END .IF $(TARGET_TYPE) == Library $(TARGET_NAME)$(LIB_EXT) : (OBJ)/{$(OBJS)} (OBJ)/{$(ADDITIONAL_OBJS)} $(OMMakefileName) @+echo Creating $@ library file $(ECHOEND) @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) (OBJ)/{$(OBJS)} (OBJ)/{$(ADDITIONAL_OBJS)} all: $(TARGET_NAME)$(LIB_EXT) $OMModelLibs .END clean: @echo Cleanup .IF $(ADDITIONAL_OBJS) != $(NULL) $(RM) (OBJ)/{$(ADDITIONAL_OBJS)} .END .IF $(TARGET_TYPE) == Library $(RMDIR) $(OBJ) $(RM) $(TARGET_NAME)$(LIB_EXT) .ELSE $(RM) $(TARGET_NAME)$(EXE_EXT) .END # # Find out something about the specific target # ## # # Define statements # ## USE_OSEDEF_H *= yes ## # # Fetch information on CPU and BSP for the selected board # ## include $(EXAMPLES_COMMON_MAKE)$/select_cpu_and_bsp.mk ## # # Signal files # ## # # Objects # ## .IF $(EXECUTABLE_FILE_TYPE) != load_module .IF $(INCLUDE_OSE_EFS) == yes OBJECTS += startefs.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start EFS .ELIF $(INCLUDE_OSE_SHELL) == yes OBJECTS += startshell.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start SHELL .END .IF $(INCLUDE_OSE_INET) == yes OBJECTS += startinet.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start INET .END .IF $(INCLUDE_OSE_PRH) == yes OBJECTS += start_prh.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start PRH .END .END OBJECTS += $(OBJS) # Error handler: .IF $(EXECUTABLE_FILE_TYPE) != load_module OBJECTS += err_hnd.o*

.END # Early Error Handler: # To be used if MMS or MMH (via PRH) .IF $(INCLUDE_OSE_MMS) == yes OBJECTS += early_error.o .ELIF $(INCLUDE_OSE_MMS) == mmh OBJECTS += early_error.o .ELIF $(INCLUDE_OSE_PRH) == yes OBJECTS += early_error.o .END ## # # Libraries # ## ## # # Contribution to architecture specific kernel # configuration. # powerpc : ospp.con # mips : krn.con # arm : osarm.con # m68000 : os68.con # ## .IF $(TARGET_ARCH) == powerpc OSPP_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSPP_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == m68000 OS68_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OS68_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == mips KRN_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(KRN_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == arm4tle OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == arm4tbe OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == sarmle OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == sarmbe OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .END (OBJ)/osarch_con_from_example.con .PRECIOUS: $(MAKEFILE:s\-f\\) $(USERCONF) $(ECHO) Create: $@$(ECHOEND) $(ECHOEMPTY) >$@ ## # # Contribution to osemain.con # ## OSEMAIN_CON_CONTRIBUTORS !:= (OBJ)/osemain_con_from_example.con $(OSEMAIN_CON_CONTRIBUTORS) (OBJ)/osemain_con_from_example.con .PRECIOUS: $(MAKEFILE:s\-f\\) $(USERCONF) $(ECHOEMPTY)>$@ $(ECHO)/* The entries below are added by makefile.mk */$(ECHOEND)>>$@ $(ECHO)/* They represent the parameters for the application. */$(ECHOEND)>>$@ .IF $(EXECUTABLE_FILE_TYPE) != load_module .IF $(INCLUDE_OSE_EFS) == yes $(ECHO)PRI_PROC(start_efs, start_efs, 1023, 9, DEFAULT, 0, NULL)$(ECHOEND)>>$@ .ELIF $(INCLUDE_OSE_SHELL) == yes $(ECHO)PRI_PROC(start_shell, start_shell, 1023, 9, DEFAULT, 0, NULL)$(ECHOEND)>>$@ .END .IF $(INCLUDE_OSE_INET) == yes $(ECHO)PRI_PROC(init_inet, init_inet, 256, 9, DEFAULT, 0, NULL)$(ECHOEND)>>$@ .END .END .IF $(INCLUDE_OSE_PRH) == yes $(ECHO)PRI_PROC(start_prh, start_prh, 256, 10, DEFAULT, 0, NULL)$(ECHOEND)>>$@ # $(ECHO)START_OSE_HOOK2(start_prh_hook) $(ECHOEND)>>$@ .END .IF $(TARGET_TYPE) == Executable $(ECHO)PRI_PROC($OMMainName, $OMMainName, 1000, 5, DEFAULT, 0, NULL) $(ECHOEND)>>$@ .END ## # # Contribution to softose.con % Softkernel environments # ## .IF $(USE_OSEDEF_H) == yes include $(EXAMPLES_COMMON_MAKE)$/osedef.mk .END ## # # Inclusion of OSE products # ## include $(EXAMPLES_COMMON_MAKE)$/products.mk # The COMPILERMAKE macro is assigned in commonsetup.mk # This has to be done late since this makefile may check things like # USE_MMS and such things that need to modify like CRT0 EXECUTABLE_NAME = $(TARGET_NAME) include $(EXAMPLES_COMMON_MAKE)$/compiler.mk LCFDEFINES += -DIMAGE_START=$(IMAGE_START) LCFDEFINES += -DIMAGE_MAX_LENGTH=$(IMAGE_MAX_LENGTH) CXXFLAGS += -Xansi include $(EXAMPLES_COMMON_MAKE)$/compilation_rules.mk $(eq,$(TARGET_TYPE),Library .EXIT: .IGNORE:) include $(EXAMPLES_COMMON_MAKE)$/targets.mk ## # IMPORT SHELL ENVIRONMENT ## # Import the environment variable PATH #.IMPORT: PATH

END OF MAKEFILE

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

*Default = "([^"]+)"[,][ ]*line ([0-9]+)[:] (error|warning) (.*)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

*Default = "([^"]+)"[,][ ]*line ([0-9]+)[:] (error|warning)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

*Default = "([^"]+)"[,][ ]*line ([0-9]+)[:] (error)*

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

*Default = "([^"]+)"[,][ ]*line ([0-9]+)[:] (warning)*

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the

machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default value = Cleared

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

OseSfk

The OseSfk metaclass contains environment settings (Compiler, framework libraries, etc.) for OSE Delta RTOS Win32 simulator compiled using Microsoft VC++ compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/OSE

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default =

receive __asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release

version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"\"$OMROOT\"\etc\Osesfkmake.bat osesfkbuild.mak buildLibs \"LIB_DIR=..\lib\" \"BUILD_SET=DEBUG\" \"USE_STL=FALSE\" \"USE_PDB=FALSE\" \"BUILD_TARGET=clean all\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)$/LangCpp $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES)

ConvertHostToIP

The ConvertHostToIP property specifies whether to convert the host name to an IP number. This is necessary because pSOSystem does not include a name service.

Default = Checked

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty MultiLine

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -DOS_DEBUG /Zi /Od /MDd /Fd"$(TARGET_NAME)"

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = /Ox /MD /Fd"$(TARGET_NAME)"

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default =

$OMFileObjPath : $OMFileImpPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property

EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = rhposemain

See also the definition of the EntryPointDeclarationModifier property for more information.

EntryPointDeclarationModifier

The EntryPointDeclarationModifier property specifies a modifier for the entry point declaration. This property allows generation of the main() function in the specified syntax.

To modify the main() signature implemented in the OSE adapter, do the following:

- Add the property EntryPointDeclarationModifier to your environment properties and set it to the main return value and name. For example: "int main"
- Set the EntryPoint property to the main arguments. For example: "int a, long b, char**"
- Generate the code.

You will get the following main() declaration:

int main(int a, long b, char** c) { ... }

Default = OS_PROCESS

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = Empty string

GeneratedAllDependencyRule

The GeneratedAllDependencyRule property specifies whether to automatically generate the “all:” rule as part of the expansion of the $OMContextMacros keyword in the makefile. If this is Cleared, you can define the makefile macros manually.

Default = Cleared

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values is as follows:

Default = .cc

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$OMROOT/etc/osesfkRun.bat" $executable

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\osesfkmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default value = .lib

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = -nologo $OMLinkCommandSet

MainIncludes

The MainIncludes property is a string that specifies the files that need to be included in the main program generated for an application.

Default = <ose.h>

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mk)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags

- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

*Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ### ###### Predefined macros ################ INCLUDE_QUALIFIER = -I LIB_CMD=$(LD) -lib LIB_FLAGS= LINK_FLAGS = $OMConfigurationLinkSwitches ### ####### Context macros ################## $OMContextMacros ## oseatexit.c: $(CP) "$(OMROOT)"\MakeTmpl\oseatexit.c oseatexit.c #.PHONY : all .DEFAULT : all LIB_PREFIX = osesfk LIB_POSTFIX = .IF $(TARGET_TYPE) == Executable OBJS += $OMFileObjPath .END .IF $(INSTRUMENTATION) == Animation INST_FLAGS=-DOMANIMATOR -GX INST_INCLUDES=$(INCLUDE_QUALIFIER) $(OMROOT)$/LangCpp$/aom $(INCLUDE_QUALIFIER) $(OMROOT)$/LangCpp$/tom INST_LIBS= $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)aomanim$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)omcomappl$(LIB_POSTFIX)$(LIB_EXT) OBJS += oseatexit.o .ELIF $(INSTRUMENTATION) == Tracing INST_FLAGS=-DOMTRACER -GX INST_INCLUDES=$(INCLUDE_QUALIFIER) $(OMROOT)$/LangCpp$/aom $(INCLUDE_QUALIFIER) $(OMROOT)$/LangCpp$/tom INST_LIBS=$(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)tomtrace$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)aomtrace$(LIB_POSTFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)oxfinst$(LIB_POSTFIX)$(LIB_EXT) $(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) OBJS += oseatexit.o .ELIF $(INSTRUMENTATION) == None INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)$/LangCpp$/lib$/$(LIB_PREFIX)oxf$(LIB_POSTFIX)$(LIB_EXT) SOCK_LIB= .ELSE MAKEFILE_ERROR = yes ERROR_TYPE = user ERROR_MSG = An invalid Instrumentation INSTRUMENTATION=$(INSTRUMENTATION) is specified. .END ## usage .PHONY: $(ECHO)Available make targets are: $(ECHOEND) $(ECHO) clean - delete the directory $(OBJ) and all its files.$(ECHOEND) $(ECHO) all - build executable file.$(ECHOEND) $(ECHOEMPTY) ## # SETS HOST TO EITHER UNIX OR WIN32 ## HOST = $(eq,$(OS),unix UNIX WIN32) ## # READ THE USER CONFIGURATION FILE ## # The USERCONF macro can be overrided on the command line. E.g. # > dmake USERCONF=~/myconf.mk all #USERCONF *= .$/userconf.mk #include $(USERCONF)*

*## # THE USER CONFIGURATION ## USERCONF *= $(OMROOT)$/MakeTmpl$/oseW32conf.mk include $(USERCONF) ## CXXFLAGS += $(ConfigurationCPPCompileSwitches) LIBRARIES += $(INST_LIBS) $(OXF_LIBS) $(LIBS) $(SOCK_LIB) ## OBJ = .$/obj OBJ_SUBDIR = .IF $(OBJ) != $(NULL) .IF $(OBJ) != . $(OBJ) .IGNORE: $(ECHO)Create: $@ $(ECHOEND) $(MKDIR) $(OBJ) .IF $(OBJ_SUBDIR) != $(NULL) $(MKDIR) $@ .END all: oseatexit.c $(OBJ) CLEAN_OBJ .PHONY: $(RMDIR) $(OBJ) CLEAN += CLEAN_OBJ .END .END SRC = . INCLUDE += -I$(OBJ) INCLUDE += -I. EXAMPLES_COMMON_CONF *= $(EXAMPLES_COMMON)$/conf EXAMPLES_COMMON_INCLUDE *= $(EXAMPLES_COMMON)$/include EXAMPLES_COMMON_MAKE *= $(EXAMPLES_COMMON)$/make EXAMPLES_COMMON_SRC *= $(EXAMPLES_COMMON)$/src INCLUDE += -I$(EXAMPLES_COMMON_INCLUDE) # Inclusion of your common settings. # In this file, you can enter constants to be used for all # examples, e.g. COMPILER, COMPILERROOT etc. include $(EXAMPLES_COMMON_MAKE)$/common_settings.mk .IF $(HOST) == UNIX include $(EXAMPLES_COMMON_MAKE)$/tools-unix.mk .ELSE include $(EXAMPLES_COMMON_MAKE)$/tools-win32.mk .END .IF $(TARGET_TYPE) == Library $(TARGET_NAME)$(LIB_EXT) : (OBJ)/{$(OBJS)} (OBJ)/{$(ADDITIONAL_OBJS)} $(OMMakefileName) @+echo Creating $@ library file $(ECHOEND) @$(LIB_CMD) $(LIB_FLAGS) /OUT:$(TARGET_NAME)$(LIB_EXT) (OBJ)/{$(OBJS)} (OBJ)/{$(ADDITIONAL_OBJS)} all: $(TARGET_NAME)$(LIB_EXT) $OMModelLibs .END clean: @echo Cleanup .IF $(ADDITIONAL_OBJS) != $(NULL) $(RM) (OBJ)/{$(ADDITIONAL_OBJS)} .END .IF $(TARGET_TYPE) == Library $(RMDIR) $(OBJ) $(RM) $(TARGET_NAME)$(LIB_EXT) .ELSE $(RM) $(TARGET_NAME)$(EXE_EXT) .END # # Find out something about the specific target # ## # # Define statements # ## USE_OSEDEF_H *= yes ## # # Fetch information on CPU and BSP for the selected board # ## include $(EXAMPLES_COMMON_MAKE)$/select_cpu_and_bsp.mk ## # # Signal files # ## # # objects # ## .IF $(EXECUTABLE_FILE_TYPE) != load_module .IF $(INCLUDE_OSE_EFS) == yes OBJECTS += startefs.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start EFS .ELIF $(INCLUDE_OSE_SHELL) == yes OBJECTS += startshell.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start SHELL .END .IF $(INCLUDE_OSE_INET) == yes OBJECTS += startinet.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start INET .END .IF $(INCLUDE_OSE_PRH) == yes OBJECTS += start_prh.o # This file is located in # <OSEROOT>/<PLATFORM>/src, and is # an example on how to start PRH .END .END OBJECTS += $(OBJS) # Error handler: .IF $(EXECUTABLE_FILE_TYPE) != load_module OBJECTS += err_hnd.o .END # Early Error Handler: # To be used if MMS or MMH (via PRH) .IF $(INCLUDE_OSE_MMS) == yes OBJECTS += early_error.o .ELIF $(INCLUDE_OSE_MMS) == mmh OBJECTS += early_error.o .ELIF $(INCLUDE_OSE_PRH) == yes OBJECTS += early_error.o .END ## # # Libraries # ## # # Contribution to architecture specific kernel # configuration. # powerpc : ospp.con # mips : krn.con # arm : osarm.con # m68000 : os68.con # ## .IF $(TARGET_ARCH) == powerpc OSPP_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSPP_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == m68000 OS68_CON_CONTRIBUTORS*

!:= (OBJ)/osarch_con_from_example.con $(OS68_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == mips KRN_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(KRN_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == arm4tle OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == arm4tbe OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == sarmle OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .ELIF $(TARGET_ARCH) == sarmbe OSARM_CON_CONTRIBUTORS !:= (OBJ)/osarch_con_from_example.con $(OSARM_CON_CONTRIBUTORS) .END (OBJ)/osarch_con_from_example.con .PRECIOUS: $(MAKEFILE:s\-f\\) $(USERCONF) $(ECHO) Create: $@$(ECHOEND) $(ECHOEMPTY) >$@ ## # # Contribution to osemain.con # ## OSEMAIN_CON_CONTRIBUTORS !:= (OBJ)/osemain_con_from_example.con $(OSEMAIN_CON_CONTRIBUTORS) (OBJ)/osemain_con_from_example.con .PRECIOUS: $(MAKEFILE:s\-f\\) $(USERCONF) $(ECHOEMPTY)>$@ $(ECHO)/* The entries below are added by makefile.mk */$(ECHOEND)>>$@ $(ECHO)/* They represent the parameters for the application. */$(ECHOEND)>>$@ .IF $(EXECUTABLE_FILE_TYPE) != load_module .IF $(INCLUDE_OSE_EFS) == yes $(ECHO)PRI_PROC(start_efs, start_efs, 1023, 9, DEFAULT, 0, NULL)$(ECHOEND)>>$@ .ELIF $(INCLUDE_OSE_SHELL) == yes $(ECHO)PRI_PROC(start_shell, start_shell, 1023, 9, DEFAULT, 0, NULL)$(ECHOEND)>>$@ .END .IF $(INCLUDE_OSE_INET) == yes $(ECHO)PRI_PROC(init_inet, init_inet, 256, 9, DEFAULT, 0, NULL)$(ECHOEND)>>$@ .END .END .IF $(INCLUDE_OSE_PRH) == yes $(ECHO)PRI_PROC(start_prh, start_prh, 256, 10, DEFAULT, 0, NULL)$(ECHOEND)>>$@ # $(ECHO)START_OSE_HOOK2(start_prh_hook) $(ECHOEND)>>$@ .END .IF $(TARGET_TYPE) == Executable $(ECHO)PRI_PROC($OMMainName, $OMMainName, 1000, 5, DEFAULT, 0, NULL) $(ECHOEND)>>$@ .END ## # # Contribution to softose.con % Softkernel environments # ## .IF $(USE_OSEDEF_H) == yes include $(EXAMPLES_COMMON_MAKE)$/osedef.mk .END ## # # Inclusion of OSE products # ## include $(EXAMPLES_COMMON_MAKE)$/products.mk # The COMPILERMAKE macro is assigned in commonsetup.mk # This has to be done late since this makefile may check things like # USE_MMS and such things that need to modify like CRT0 EXECUTABLE_NAME = $(TARGET_NAME) include $(EXAMPLES_COMMON_MAKE)$/compiler.mk LCFDEFINES += -DIMAGE_START=$(IMAGE_START) LCFDEFINES += -DIMAGE_MAX_LENGTH=$(IMAGE_MAX_LENGTH) include $(EXAMPLES_COMMON_MAKE)$/compilation_rules.mk $(eq,$(TARGET_TYPE),Library .EXIT: .IGNORE:) include $(EXAMPLES_COMMON_MAKE)$/targets.mk ## # IMPORT SHELL ENVIRONMENT ## # Import the environment variable PATH #.IMPORT: PATH ## # END OF MAKEFILE ##

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (warning)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default value = Cleared

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output

window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

Package

The Package metaclass contains properties that affect packages.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to Cleared, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to Cleared, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to Cleared, all the arguments are not animated.
- If the AnimateArguments property is set to Cleard, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = Checked

ContributesToNamespace

The ContributesToNamespace property specifies whether the packages contained in this package is declared as children packages of this package. Regardless of the setting, a directory is created for the current package to hold its contained elements. (Default = True)

DefineNameSpace

The DefineNameSpace property specifies whether a package defines a namespace. A namespace is a declarative region that attaches an additional identifier to any names declared inside it.

Default = Cleared

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table: Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types
- $Tag - The value of the specified element’s tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

EventsBaseID

The EventsBaseID property specifies the base ID for events. The default values for the C++ Package environment is "1".

GenerateDirectory

The GenerateDirectory property specifies whether to generate a separate directory for the package.

The possible values are as follows:

- Checked - The package generates a directory.
- Cleared - The package will not generate a directory. (This is the default.)

GenerateDirectory has an immediate effect on directory generation.

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces.

Default = Empty string

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body.

Default = Empty MultiLine

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body.

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

InitializationCode

The InitializationCode property specifies the user-defined initialization code to add to the package body. (empty MultiLine)

IsNested

The IsNested property specifies whether to generate the class or package as nested. (Default = False)

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private. (Default = False)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

Default = None

NameSpaceName

By default, if you have set the property CPP_CG::Package::DefineNameSpace to True, the name used for the namespace is the name of the package. The property NameSpaceName allows you to specify a different name to use for the namespace.

Default = Blank

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package.

Default = Public

PackageClassNamePolicy

The PackageClassNamePolicy property specifies the naming policy for classes generated by Rational

Rhapsody. Rhapsody generates a class for each package in the Rational Rhapsody Developer for Java model. The possible values are as follows:

- Default - Use the default naming style (the package class name is the same as the package name).
- WithSuffix - Add a suffix to the class name. The suffix is “_pkgClass”.

Default = Default

PackageEventIdRange

The PackageEventIdRange property specifies the maximum number of events allowed in a package. This property is set on the component level.

Default = 200

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

Default = Empty string

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification.

Default = Empty MultiLine

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification.

Default = Empty MultiLine

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces.

Default = Empty string

Port

The Port metaclass controls whether code is generated for ports.

Generate

The Generate property specifies whether to generate code for a particular type of element.

Default = Checked

HandleDisconnectedPort

The property HandleDisconnectedPort allows you to provide a code fragment that handles cases where the link of a port is not initialized.

The following is an example of such code:

```
short eventID = $RuntimeEventID;

if (eventID == -1)

{

cout << "Warning: Operation $OpName was not sent via port $PortOwnerName:$PortName since link via interface $InterfaceName is not initialized\n" << endl;

}

else

{

cout << "Warning: Event ID " << eventID << " was not sent via port $PortOwnerName:$PortName since link via interface $InterfaceName is not initialized\n" << endl;

}
```

The following keywords can be used in your code:

$OpName - the name of the primitive operation being called via the port

$RuntimeEventID - the event ID of the event that was sent to the port. In the case of primitive operations, -1 is used as the value of this keyword.

$PortName - the name of the port

$PortOwnerName - the name of the class or object that owns the port

$InterfaceName - the interface that declared the service requested via the port

Note that this property is available only for explicit (non-rapid) ports, and only for C++.

Default = Empty

HandleUnknownEvent

The property HandleUnknownEvent allows you to provide a code fragment that handles cases where an unknown event is sent via the port.

The following is an example of such code:

```
cout <<"Event ID " << $RuntimeEventID << " is not recognized in port $PortOwnerName:$PortName"
```

<< endl;

The following keywords can be used in your code:

$OpName - the name of the primitive operation being called via the port

$RuntimeEventID - the event ID of the event that was sent to the port. In the case of primitive operations, -1 is used as the value of this keyword.

$PortName - the name of the port

$PortOwnerName - the name of the class or object that owns the port

$InterfaceName - the interface that declared the service requested via the port

Note that this property is available only for explicit (non-rapid) ports, and only for C++.

Default = Empty

UseExactTypeForReqPureReactiveInterface

The property UseExactTypeForReqPureReactiveInterface determines whether a port that only has pure reactive required interfaces (interfaces that only have event receptions) can be connected to a rapid port (port with no explicit contract - relays any kind of event) or only to a port that provides the required interface. The possible values are:

- False - The port can be connected to a rapid port.
- True - The port can only be connected to a port that provides the required interface.

Default = False

Note that this property is only relevant for ports that only have pure reactive required interfaces.

QNXNeutrinoMomentics

The QNXNeutrinoMomentics metaclass contains environment settings (Compiler, framework libraries, etc.) for QNX Neotrino RTOS compiled for X86, using GCC cross compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/QNX

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"\"$OMROOT\"\etc\qnxcwmake.bat qnxcwbuild.mak build \"CPU=$CPU\" \"CPU_SUFFIX=$CPU_SUFFIX\" \"

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp

-I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

The default value is as follows: -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)\LangCpp\lib\QNXCWWebComponents$(CPU)$(CPU_SUFFIX)$(LIB_EXT), $(OMROOT)\lib\QNXCWWebServices$(CPU)$(CPU_SUFFIX)$(LIB_EXT), -lsocket

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated

implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\qnxcwmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -O

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = -static

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ### ###### Predefined macros ################ RM=rm -rf MD=mkdir -p INCLUDE_QUALIFIER=-I CPU=$OMCPU CPU_SUFFIX=$OMCPU_SUFFIX CC=qcc -Vgcc_nto(CPU)(CPU_SUFFIX) -I$(QNX_TARGET)/usr/include -lang-c++ -DUSE_IOSTREAM LIB_CMD=$(QNX_HOST)/usr/gcc/nto$(CPU)/bin/ar LINK_CMD=$(CC) LIB_FLAGS=rvu LINK_FLAGS=-static ### ####### Context macros ################## $OMContextMacros ### ####### Predefined macros ############### $(OBJS) : $(INST_LIBS) $(OXF_LIBS) OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= $(MD) $(OBJ_DIR) CLEAN_OBJ_DIR= $(RM) $(OBJ_DIR) endif ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/QNXCWaomanim(CPU)(CPU_SUFFIX)$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/QNXCWoxfinst(CPU)(CPU_SUFFIX)$(LIB_EXT) $(OMROOT)/LangCpp/lib/QNXCWomcomappl(CPU)(CPU_SUFFIX)$(LIB_EXT) SOCK_LIB=-lsocket else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/QNXCWtomtrace(CPU)(CPU_SUFFIX)$(LIB_EXT) $(OMROOT)/LangCpp/lib/QNXCWaomtrace(CPU)(CPU_SUFFIX)$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/QNXCWoxfinst(CPU)(CPU_SUFFIX)$(LIB_EXT) $(OMROOT)/LangCpp/lib/QNXCWomcomappl(CPU)(CPU_SUFFIX)$(LIB_EXT) SOCK_LIB=-lsocket else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/QNXCWoxf(CPU)(CPU_SUFFIX)$(LIB_EXT) SOCK_LIB= else @echo An invalid Instrumentation $(INSTRUMENTATION) is specified. exit endif endif .SUFFIXES: $(CPP_EXT) ## ###################### Context dependencies and commands ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @$(CC) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## # Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) -o $(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS $(RM) $OMFileObjPath $(ADDITIONAL_OBJS) $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OMCPU

The OMCPU property is resolved in the MakeFileContent property as the CPU type. The QNXNeutrinoCW environment uses the custom keywords feature to enable you to select the CPU without

modifying the makefile template.

Default = x86

OMCPU_SUFFIX

The OMCPU_SUFFIX property is resolved in the MakeFileContent property as the CPU extension (which is required for PPC targets). The QNXNeutrinoCW environment uses the custom keywords feature to enable you to select the CPU without modifying the makefile template.

Default = ($NO_CPU_EXT)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (make)(.)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Cleared

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Checked

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

QNXNeutrinoGCC

The QNXNeutrinoGCC metaclass contains environment settings (Compiler, framework libraries, etc.) for QNX Neotrino RTOS compiled for X86, using GCC on-target compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.

- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/QNX

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release

version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

The default value is as follows: -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)\LangCpp\lib\QNXWebComponents$(LIB_EXT),
$(OMROOT)\lib\QNXWebServices$(LIB_EXT), -lsocket

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = $executable

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = Empty string

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -O

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ##################
##
CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease

LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ###### Predefined macros ################ RM=/bin/rm -rf MD=/bin/mkdir -p INCLUDE_QUALIFIER=-I CC=gcc -I/usr/include -DUSE_IOSTREAM LIB_CMD=ar LINK_CMD=$(CC) LIB_FLAGS=rvu LINK_FLAGS= /x86/lib/libm.so.1 -lstdc++ $OMConfigurationLinkSwitches ## ####### Context macros ################## $OMContextMacros ## ####### Predefined macros ############### $(OBJS) : $(INST_LIBS) $(OXF_LIBS) OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= $(MD) $(OBJ_DIR) CLEAN_OBJ_DIR= $(RM) $(OBJ_DIR) endif ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/QNXaomanim$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/QNXoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/QNXomcomappl$(LIB_EXT) SOCK_LIB=-lsocket else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/QNXtomtrace$(LIB_EXT) $(OMROOT)/LangCpp/lib/QNXaomtrace$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/QNXoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/QNXomcomappl$(LIB_EXT) SOCK_LIB=-lsocket else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/QNXoxf$(LIB_EXT) SOCK_LIB= else @echo An invalid Instrumentation $(INSTRUMENTATION) is specified. exit endif endif endif .SUFFIXES: $(CPP_EXT) ## ###################### Context dependencies and commands ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @$(CC) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## # Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) -o $(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS $(RM) $OMFileObjPath $(ADDITIONAL_OBJS) $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated

specification (header) files for a given language and environment.

Default = .h

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Checked

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

Relation

The Relation metaclass contains properties that affect relations.

Add

The Add property specifies the command used to add an item to a container.

Default = Add_$target:c

AddGenerate

The AddGenerate property specifies whether to generate an Add() operation for relations. (Default = True)

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CPP_CG::Type::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

Default = Checked

Clear

The Clear property specifies the name of an operation that removes all items from a relation.

Default = Clear_$target:c

ClearGenerate

The ClearGenerate property specifies whether to generate a Clear() operation for relations. (Default = True)

CreateComponent

The CreateComponent property specifies the name of an operation that creates a new component in a composite class.

Default = New_$target:c

CreateComponentGenerate

The CreateComponentGenerate property specifies whether to generate a CreateComponent operation for composite objects. Setting this property to False is one way to optimize your code for size. (Default = True)

DataMemberVisibility

The DataMemberVisibility property specifies the visibility of the relation data member. For example, if the relation is implemented as a pointer, this property determines whether the pointer data member is declared as public, private, or protected. The default value for C++ is Protected.

DeleteComponent

The DeleteComponent property specifies the name of an operation that deletes a component from a composite class.

Default = Delete_$target:c

DeleteComponentGenerate

The DeleteComponentGenerate property specifies whether to generate a DeleteComponent() operation for composite objects. (Default = True)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation

rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table: Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument's description Operation Primitive operations, triggered operations, $Arguments - The operation argument's description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types
- $Tag - The value of the specified element's tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

Find

The Find property specifies the name of an operation that locates an item among relational objects.

Default = Find_$target:c

FindGenerate

The FindGenerate property specifies whether to generate a Find() operation for relations. (Default = False)

Get

The Get property specifies the name of an operation that retrieves the relation currently pointed to by the iterator.

Default = Get_$target:c

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index. The ContainerTypes::Relationtype::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

Default = get$cname:cAt

GetAtGenerate

The GetAtGenerate property specifies whether to generate a getAt() operation for relations. The possible values are as follows:

- Checked - Generate a getAt() operation for relations.
- Cleared - Do not generate a getAt() operation for relations. Setting the GetAtGenerate property to False is one way to optimize your code for size.

Default = Cleared

GetEnd

The GetEnd property specifies the name of an operation that points the iterator to the last item in a collection.

Default = Get_$target:cEnd

GetEndGenerate

The GetEndGenerate property specifies whether to generate a GetEnd() operation for relations. (Default = True)

GetGenerate

The GetGenerate property specifies whether to generate accessor operations for relations. (Default = True)

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key. For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default = get$cname:c

GetKeyGenerate

The GetKeyGenerate property specifies whether to generate getKey() operations for relations. Setting this property to False is one way to optimize your code for size.

Default = Checked

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

Default = Empty MultiLine

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

Default = Empty MultiLine

ImplementWithStaticArray

The ImplementWithStaticArray property specifies whether to implement relations as static arrays. The possible values are as follows:

- Default - Rational Rhapsody provides the appropriate implementation for all fixed and bounded relations.
- FixedAndBounded - All fixed and bounded relations are generated into static arrays.

To generate C-like code in C++ or Java, modify the value of the ImplementWithStaticArray property to FixedAndBounded.

Default = FixedAndBounded

InitializeComposition

The InitializeComposition property controls how a composition relation is initialized. The possible values are as follows:

- InInitializer
- InRecordType
- None

Default = InInitializer

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for C++ The possible values for the Inline property are as follows:

- none - The operation is not generated inline.
- in_header - The operation is generated inline in the specification file.
- in_source - The operation is generated inline in the implementation file.
- in_declaration - A class operation is generated inline in the class declaration. A global function is generated inline in the package specification file.

Inlining an operation in the header might cause problems if the function body refers to other classes. For example, if the inlined code refers to another class (via a pointer such as itsRelatedClass), inlined code generated in a header might not compile. The implementation file for the class would have an #include for

RelatedClass, but the specification file would not. The workaround is to create a Usage dependency of the class with the inlined function on the related class. This forces an #include of the related class to be generated in the header of the dependent class with the inlined function.

Default = none

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased. (Default = False)

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

ObjectInitialization

The ObjectInitialization property specifies what kind of initialization will occur for the initial instances of a configuration. The possible values are as follows:

- Full - Instances are initialized and their behavior is started.
- Creation - Instances are initialized but their behavior is not started.
- None - Instances are not initialized and their behavior is not started.

Default = Full

Remove

The Remove property specifies the name of an operation that removes an item from a relation.

Default = Remove_$target:c

RemoveGenerate

The RemoveGenerate property specifies whether to generate a Remove() operation for relations. (Default = True)

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = remove$cname:c

RemoveKeyGenerate

The RemoveKeyGenerate property specifies whether to generate a removeKey() operation for qualified relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

RemoveKeyHelpersGenerate

The RemoveKeyHelpersGenerate property enables you to control the generation of the relation helper methods (for example, _removeItsX() and __removeItsX()). The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the CPP_CG::Relation::RemoveKey property.

(Default = True)

SafeInitScalar

The SafeInitScalar property specifies whether to initialize scalar relations as null pointers. (Default = False)

Set

The Set property specifies the name of the mutator generated for scalar relations.

Default = Set_$target:c

SetGenerate

The SetGenerate property specifies whether to generate mutators for relations. (Default = True)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to "abstract." You must include the space after the word "abstract." If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class Yes No Inside Package Yes Yes Inside

(empty MultiLine)

Static

The Static property is a Boolean value that determines whether class-wide relations are enabled. Class-wide members of a class are shared between all instances of that class and are mapped as static. When a relation is tagged as static:

- The data member is generated as static (with the static keyword).
- The relation accessors are generated as static.
- The mutators of directional relations are generated as static. The mutators of symmetric relations are generated as common (non-static) operations.

Note the following behavior and restrictions:

- If there are links between instances based on static relations, code generation will initialize all the valid links. In case of a limited relation size, the last initialization is preserved.
- When you generate instrumented code (animation or tracing), relation NOTIFY calls are not added to static relation mutators.
- Animation associates static relations with the class instances, not the class itself.
- In an instrumented application (animation or tracing), the static relations names appear in each instance node; however, the values of directional static relations are not visible.

See also the properties CG::Relation::Containment, Containertype::Relationtype::CreateStatic, and Containertype::Relationtype::InitStatic.

Default = Cleared

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language.

Default = Public

Solaris2

The Solaris2 metaclass contains environment settings (Compiler, framework libraries, etc.) for Solaris 2, using Sun compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/Solaris2

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects

the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a

debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line

number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default =ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the

Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default is as follows:

$(OMROOT)\LangCpp\lib\sol2WebComponents$(LIB_EXT),
$(OMROOT)\lib\sol2WebServices$(LIB_EXT), -lsocket -lnsl

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = xterm -e $executable

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = $OMROOT/etc/sol2make $makefile $maketarget

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -O

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ###### Predefined macros ################ RM=/bin/rm -rf MD=/bin/mkdir -p INCLUDE_QUALIFIER=-I CC=gcc -I/usr/include -DUSE_IOSTREAM LIB_CMD=ar LINK_CMD=$(CC) LIB_FLAGS=rvu LINK_FLAGS= -lposix4 -lpthread -lstdc++ $OMConfigurationLinkSwitches ### ######## Context macros ################## $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= $(MD) $(OBJ_DIR) CLEAN_OBJ_DIR= $(RM) $(OBJ_DIR) endif ### ######## Predefined macros ############### $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/sol2aomanimGNU$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/sol2oxfinstGNU$(LIB_EXT) $(OMROOT)/LangCpp/lib/sol2omcomapplGNU$(LIB_EXT) SOCK_LIB=-lsocket -lnsl else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/sol2tomtraceGNU$(LIB_EXT) $(OMROOT)/LangCpp/lib/sol2aomtraceGNU$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/sol2oxfinstGNU$(LIB_EXT) $(OMROOT)/LangCpp/lib/sol2omcomapplGNU$(LIB_EXT) SOCK_LIB=-lsocket -lnsl else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/sol2oxfGNU$(LIB_EXT) SOCK_LIB= else @echo An invalid Instrumentation $(INSTRUMENTATION) is specified. exit endif endif .SUFFIXES: $(CPP_EXT) ## ################### Context dependencies and commands ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @$(CC) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## #

Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries #
##
$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) -o $(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS $(RM) $OMFileObjPath $(ADDITIONAL_OBJS) $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Checked

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

Solaris2GNU

The Solaris2GNU metaclass contains environment settings (Compiler, framework libraries, etc.) for Solaris 2, using GCC compiler .

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/Solaris2

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects

the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a

debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = -O

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

Default =

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default value = main

If applicable, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line

number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the

Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default is as follows:

$(OMROOT)\LangCpp\lib\sol2WebComponentsGNU$(LIB_EXT),
$(OMROOT)\lib\sol2WebServicesGNU$(LIB_EXT),-lsocket -lnsl

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = xterm -e $executable

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = $OMROOT/etc/sol2make $makefile $maketarget

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -O

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ###### Predefined macros ################ RM=/bin/rm -rf MD=/bin/mkdir -p INCLUDE_QUALIFIER=-I TMPL_DIR=./Tmpl$(TARGET_NAME) CACHE_DIR=./SunWS_cache CC=CC -mt -ptr$(TMPL_DIR) LIB_CMD=$(CC) LINK_CMD=$(CC) LIB_FLAGS=-xar $OMConfigurationLinkSwitches LINK_FLAGS= -lposix4 -lpthread $OMConfigurationLinkSwitches ## ####### Context macros ################## $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= $(MD) $(OBJ_DIR) CLEAN_OBJ_DIR= $(RM) $(OBJ_DIR) endif ## ####### Predefined macros ############### $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/sol2aomanim$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/sol2oxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/sol2omcomappl$(LIB_EXT) SOCK_LIB= -liostream -lsocket -lintl -lnsl -lCrun -lCstd else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/sol2tomtrace$(LIB_EXT) $(OMROOT)/LangCpp/lib/sol2aomtrace$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/sol2oxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/sol2omcomappl$(LIB_EXT) SOCK_LIB= -liostream -lsocket -lintl -lnsl -lCrun -lCstd else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/sol2oxf$(LIB_EXT) SOCK_LIB= else @echo An invalid Instrumentation $(INSTRUMENTATION) is specified. exit endif endif endif .SUFFIXES: $(CPP_EXT) ## ###################### Context dependencies and commands ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @$(CC) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile

Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) -o $(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) -o $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS $(RM) $OMFileObjPath $(ADDITIONAL_OBJS) $(RM) $(TMPL_DIR) $(CACHE_DIR) $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Checked

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = "([^\"]+)"[,][ ]line ([0-9]+)[:] (Error|Warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Checked

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

Statechart

The Statechart metaclass contains the statechart code generation properties.

StatechartImplementation

Prior to version 7.3 of Rational Rhapsody, the transition-handling code generated by Rational Rhapsody used a switch statement to represent the possible states. Beginning with version 7.3, this code uses an if/else structure. To allow older models to use the previous code generation behavior, a property called StatechartImplementation was added to the Pre73 backward compatibility profiles. The possible values for the property are:

- SwitchOnly - transition-handling code uses a switch statement to represent the possible states
- Default - the transition-handling code uses an if/else structure to represent the possible states

Default = SwitchOnly

StatechartStateOperations

The StatechartStateOperations property controls the state operations in a statechart.

Default = None

Type

The Type metaclass contains a property that affects the visibility of data types.

AnimEnumerationTypeImage

The AnimEnumerationTypeImage property is a Boolean value that determines whether the Image attribute is used for enumerated types when using animation.

Default = False

AnimSerializeOperation

The AnimSerializeOperation property enables you to specify the name of an external function used to animate all attributes and arguments that are of that type. Rational Rhapsody can animate (display) the values of simple types and one-dimensional arrays without any problem. To display the current values of such attributes during an animation session, run the features window for the instance. However, if you want to animate a more complex type, such as a date, the type must be converted to a string (char *) for Rational Rhapsody to display it. This is generally done by writing a global function, an instrumentation function, that takes one argument of the type you want to display, and returns a char *. You must disable animation of the instrumentation function itself (using the Animate and AnimateArguments properties for the function). For example, you can have a type tDate, defined as follows:

typedef struct date { int day; int month; int year; } %s;

You can have an object with an attribute count of type int, and an attribute date of type tDate. The object can have an initializer with the following body:

me-date.month = 5; me-date.day = 12; me-date.year = 2000; If you want to animate the date attribute, the AnimSerializeOperation property for date must be set to the name of a function that will convert the type tDate to char *. For example, you can set the property to a function named showDate. This function name must be entered without any parentheses. It must take an attribute of type tDate and return a char *. The Animate and AnimateArguments properties for the showDate function must be set to False. The implementation of the showDate function might be as follows: showDate(tDate aDate) { char* buff; buff = (char*) malloc(sizeof(char) * 20); sprintf(buff,"%d %d %d", aDate.month,aDate.day,aDate.year); return buff; }

When you run this model with animation, instances of this object will display a value of 5 12 2000 for the date attribute in the browser. If the showDate function is defined in the same class that the attribute belongs to and the function is not static, the AnimSerializeOperation property value should be similar to the following: myReal-showDate This value shows that the function is called from the serializeAttributes function, located in the class OMAnimatedclassname. The showDate function must allocate memory for the returned string via the malloc/alloc/calloc function in C, or the new operator in C++. Otherwise, the system will crash.

Default = Empty string

AnimUnserializeOperation

The AnimUnserializeOperation property converts a string to the value of an element (the opposite of the AnimSerializeOperation property). Unserialize functions are used for event generation or operation invocation using the Animation toolbar to convert the string (received from the user) to the value of the event or operation before the event generation or operation invocation. For example, your serialization operation might look similar to the following:

char* myX2String(const Rec f) { char* cS = new char[OutputStringLength]; /* conversion from the Rec type to string */ return (cS); } The unserialization operation would be: Rec myString2X (char* C, Rec T) { T = new Trc; /* conversion of the string C to the Rec type */ delete C; return (T); }

Default = Empty string

DeclarationPosition

The DeclarationPosition property specifies where the type declaration appears. The possible values are as follows:

- BeforeClassRecord - The type declaration appears before the class record (CR) declaration if CR has a visibility set to public, and before the class record forward declaration if CR has a visibility set to private.
- AfterClassRecord - The type declaration appears after the class record declaration if CR has a visibility set to public, and after the class record forward declaration if CR has a visibility set to private.
- StartOfDeclaration - The type declaration appears among the first declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the first declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.
- EndOfDeclaration - The type declaration appears among the last declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the last declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.

If the CPP_CG::Type::Visibility property is set to "Body", no matter the settings of CPP_CG::Type::DeclarationPosition property, the type declaration still appears in the package body.

Default = BeforeClassRecord

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)

- $Description - The element description

Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

Default = Empty string

EnumerationAsTypedef

The EnumerationAsTypedef property specifies whether the generated enum should be wrapped by a typedef. This property is applicable to enumeration types in C and C++.

Default = Cleared

In

The In property specifies how code is generated when the type is used with an argument that has the modifier "In".

Default = const $type&

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier "InOut".

Default = $type&

IsLimited

The IsLimited property determines whether the class or record type is generated as limited.

Default = False

LanguageMap

The LanguageMap property specifies the Ada declaration for Rational Rhapsody language-independent types.

Default = Empty string

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out".

Default = $type&*

PrivateName

The PrivateName property specifies the pattern used to generate names of private operations in C.

Default = $typeName

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C.

Default = $objectName_$typeName

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the "Reference" option for attribute/typedefs (composite types) is mapped to code.

*Default = **

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type.

*Default = $type**

StructAsTypedef

The StructAsTypedef property specifies whether the generated struct should be wrapped by a typedef. This property is applicable to structure types in C and C++.

Default = Cleared

TriggerArgument

The TriggerArgument property is used for mapping event and triggered operation arguments to code instead of the In, InOut, and Out properties. A different property is required because of code generation limitations related to event arguments. See also:

- In
- InOut
- Out

Default = $type

UnionAsTypedef

The UnionAsTypedef property specifies whether the generated union should be wrapped by a typedef. This property is applicable to union types in C and C++.

Default = Cleared

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

Default = Public

VxWorks

The VxWorks metaclass contains environment settings (Compiler, framework libraries, etc.) for VxWorks

5.4.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs \"CPU=$BSP\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

Default = CXX=$AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -O0 -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = vxmain

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(LIB_EXT),
$(OMROOT)/lib/vxWebServices(CPU)(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $OMROOT/DLLs/TornadoIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vxmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ######### Definitions and flags ################## ## CPU = $BSP TOOL = gnu include $(WIND_BASE)/target/h/make/defs.bsp .cpp.o : @ $(RM) $@ $(CXX) $(C++FLAGS) $(OPTION_OBJECT_ONLY) $< $OMCodeTestSettings INCLUDE_QUALIFIER=-I LIB_CMD=$(AR) LINK_CMD=$(LD) LIB_FLAGS=$(ARFLAGS) #LINK_FLAGS=$OMConfigurationLinkSwitches -r

$(LDFLAGS) LINK_FLAGS=$OMConfigurationLinkSwitches -r ## ################ Context generated macros ########### $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) rmdir $(OBJ_DIR) endif ## ################ Predefined macros ################### $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(LIB_EXT) SOCK_LIB= else echo 'An invalid Instrumentation $(INSTRUMENTATION) is specified.' exit endif endif ## ################## Context generated dependencies ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @echo Compiling $OMMainImplementationFile @$(CXX) $(C++FLAGS) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## # Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking and Munching $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $(LINK_FLAGS) -o $(TARGET_NAME).tmp \ $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) @ $(RM) $(TARGET_NAME)$(EXE_EXT) ctdt.c ctdt.o @$(NM) $(TARGET_NAME).tmp | $(MUNCH) > ctdt.c @$(CC) -c ctdt.c @$(LD) -r $OMLinkCommandSet -o $@ $(TARGET_NAME).tmp ctdt.o @ $(RM) ctdt.c ctdt.o $(TARGET_NAME).tmp $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: cleanall: clean @echo Cleanup $(RM) $OMFileObjPath $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $OMCleanOBJS $(CLEAN_OBJ_DIR)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileO

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (make)(.)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

VxWorks6diab

The VxWorks6diab metaclass contains settings (Compiler, framework libraries, etc.) for VxWorks 6.x, using WindRiver-Compiler compiler (previously named "Diab").

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=diab\" \"TOOL_FAMILY=diab\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

Default = CXX = $(AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

$IgnoreSwitches -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = vxmain

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression

- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(TOOL)$(LIB_EXT),
$(OMROOT)/lib/vxWebServices(CPU)(TOOL)$(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -X -r

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -X -r

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you

would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ######### Definitions and flags ################## ## CPU = $BSP TOOL = diab include $(WIND_BASE)/target/h/make/defs.bsp .cpp.o : @ $(RM) $@ $(CXX) $(C++FLAGS) $(OPTION_OBJECT_ONLY) $< $OMCodeTestSettings INCLUDE_QUALIFIER=-I LIB_CMD=$(AR) LINK_CMD=$(LD) $(CC_ARCH_SPEC) LIB_FLAGS=$(ARFLAGS) LINK_FLAGS=$OMConfigurationLinkSwitches -r5 ## ############### Context generated macros ########### $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) rmdir $(OBJ_DIR) endif ## ############### Predefined macros ################## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(TOOL)$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM

INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(TOOL)$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else echo 'An invalid Instrumentation $(INSTRUMENTATION) is specified.' exit endif endif ## ################## Context generated dependencies ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @echo Compiling $OMMainImplementationFile @$(CXX) $(C++FLAGS) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## # Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking and Munching $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $(LINK_FLAGS) -o $(TARGET_NAME).tmp \ $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) @ $(RM) $(TARGET_NAME)$(EXE_EXT) ctdt.c ctdt.o @$(NM) $(TARGET_NAME).tmp | $(MUNCH) > ctdt.c @$(CC) $(CC_ARCH_SPEC) -c ctdt.c @$(LINK_CMD) $OMLinkCommandSet -r4 -o $@ $(TARGET_NAME).tmp ctdt.o @ $(RM) ctdt.c ctdt.o $(TARGET_NAME).tmp $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: cleanall: clean @echo Cleanup $(RM) $OMFileObjPath $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $OMCleanOBJS $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and

(Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (warning)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in

double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

VxWorks6diab_RTP

The VxWorks6diab_RTP metaclass contains environement settings (Compiler, framework libraries, etc.).

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.

- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=diab\" \"TOOL_FAMILY=diab\" \"BUILD=$BuildCommandSet\" \"USE_RTP=TRUE\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

Default = CXX = $(AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

$IgnoreSwitches -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .vxe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT),
$(OMROOT)/lib/vxWebServices(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP diab"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = Empty string

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the

property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ######### Definitions and flags ################## ### CPU = $BSP TOOL = diab RHP_LIB_EXT = $OMLibExt RTP_SUFFIX = _RTP_ LINK_CMD=$(CXX) $(CC_ARCH_SPEC) -Xansi -Xforce-declarations -Xmake-dependency=0xd LINK_FLAGS=$OMConfigurationLinkSwitches RTP_LIBS = -L$(WIND_USR_LIB_PATH) -lstlstd $OMCodeTestSettings INCLUDE_QUALIFIER=-I LIB_CMD=$(AR) LIB_FLAGS=$(ARFLAGS) ## ################ Context generated macros ########### $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) rmdir $(OBJ_DIR) endif ## ################ Predefined macros ################## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) SOCK_LIB=$(RTP_LIBS) else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom

INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) SOCK_LIB=$(RTP_LIBS) else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) SOCK_LIB=$(RTP_LIBS) else echo 'An invalid Instrumentation $(INSTRUMENTATION) is specified.' exit endif endif ## ################### Context generated dependencies ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @echo Compiling $OMMainImplementationFile @$(CXX) $(C++FLAGS) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## # Predefined linking instructions # # INST_LIBS is included twice to solve bi-directional dependency between libraries # ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $(LINK_FLAGS) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) -o $@ $(TARGET_NAME)$(RHP_LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(RHP_LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: cleanall: clean @echo Cleanup $(RM) $OMFileObjPath $(RM) $(TARGET_NAME)$(RHP_LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $OMCleanOBJS $(CLEAN_OBJ_DIR) include $(WIND_USR)/make/rules.rtp

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (warning)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

VxWorks6gnu

The VxWorks6gnu metaclass contains environement settings (Compiler, framework libraries, etc.) for VxWorks 6.x, using GNU compiler (GCC).

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.

- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=gnu\" \"TOOL_FAMILY=gnu\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports

integration with Applied Microsystems Corporation CodeTest.

Default = CXX = $(AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I$OMDefaultSpecificationDirectory -fno-merge-templates -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -O0 -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = vxmain

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(TOOL)$(LIB_EXT),
$(OMROOT)/lib/vxWebServices(CPU)(TOOL)$(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ######### Definitions and flags ################## ## CPU = $BSP TOOL = gnu include $(WIND_BASE)/target/h/make/defs.bsp .cpp.o : @ $(RM) $@ $(CXX) $(C++FLAGS) $(OPTION_OBJECT_ONLY) $< $OMCodeTestSettings INCLUDE_QUALIFIER=-I LIB_CMD=$(AR) LINK_CMD=$(LD) LIB_FLAGS=$(ARFLAGS) LINK_FLAGS=$OMConfigurationLinkSwitches -r ## ############### Context generated macros ########### $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) rmdir $(OBJ_DIR) endif ## ################ Predefined macros ################## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(TOOL)$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(TOOL)$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else echo 'An

invalid Instrumentation $(INSTRUMENTATION) is specified.' exit endif endif endif ## ################## Context generated dependencies ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @echo Compiling $OMMainImplementationFile @$(CXX) $(C++FLAGS) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## ############ Predefined linking instructions ##################### $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking and Munching $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $(LINK_FLAGS) -o $(TARGET_NAME).tmp \ $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) @ $(RM) $(TARGET_NAME)$(EXE_EXT) ctdt.c ctdt.o @$(NM) $(TARGET_NAME).tmp | $(MUNCH) > ctdt.c @$(CC) $(CC_ARCH_SPEC) -c ctdt.c @$(LINK_CMD) -r $OMLinkCommandSet -o $@ $(TARGET_NAME).tmp ctdt.o @ $(RM) ctdt.c ctdt.o $(TARGET_NAME).tmp $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: cleanall: clean @echo Cleanup $(RM) $OMFileObjPath $(RM) $(TARGET_NAME)$(LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $OMCleanOBJS $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated

makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error|warning)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the

machine running the application.

To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

VxWorks6gnu_RTP

The VxWorks6gnu_RTP metaclass contains environement settings (Compiler, framework libraries, etc.).

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=gnu\" \"TOOL_FAMILY=gnu\" \"BUILD=$BuildCommandSet\" \"USE_RTP=TRUE\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

Default = CXX = $(AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default =

-I$OMDefaultSpecificationDirectory -fno-merge-templates -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -O0 -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .vxe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Cleared

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Empty string

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -g

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = Empty string

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet -MD -MP

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = ############# Target type (Debug/Release) ################## ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ## ######### Definitions and flags ################## ## CPU = $BSP TOOL = gnu RHP_LIB_EXT = $OMLibExt RTP_SUFFIX = _RTP_ RTP_LIBS = -L$(WIND_USR_LIB_PATH) -lstdc++ INCLUDE_QUALIFIER=-I LIB_CMD=$(AR) LIB_FLAGS=$(ARFLAGS) LINK_CMD=$(CXX) $(CC_ARCH_SPEC) LINK_FLAGS=$OMConfigurationLinkSwitches $OMCodeTestSettings ## ################ Context generated macros ########### $OMContextMacros OBJ_DIR=$OMObjectsDir ifeq ($(OBJ_DIR),) CREATE_OBJ_DIR= CLEAN_OBJ_DIR= else CREATE_OBJ_DIR= if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) rmdir $(OBJ_DIR) endif ## ################ Predefined macros ################## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) SOCK_LIB=$(RTP_LIBS) else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) SOCK_LIB=$(RTP_LIBS) else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT) SOCK_LIB=$(RTP_LIBS) else echo 'An invalid Instrumentation $(INSTRUMENTATION) is specified.' exit endif endif ## ################## Context generated dependencies ############# $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) @echo Compiling

$OMMainImplementationFile @$(CXX) $(C++FLAGS) $(ConfigurationCPPCompileSwitches) -o $OMFileObjPath $OMMainImplementationFile ## ############ Predefined linking instructions ########################## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) @$(LINK_CMD) $(LINK_FLAGS) $(C++FLAGS) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(INST_LIBS) \ $(SOCK_LIB) -o $@ $(TARGET_NAME)$(RHP_LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(RHP_LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: cleanall: clean @echo Cleanup $(RM) $OMFileObjPath $(RM) $(TARGET_NAME)$(RHP_LIB_EXT) $(RM) $(TARGET_NAME)$(EXE_EXT) $OMCleanOBJS $(CLEAN_OBJ_DIR) include $(WIND_USR)/make/rules.rtp

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error|warning)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ([^:]+)[:]([0-9]+)[:]

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

WorkbenchManaged

The WorkbenchManaged metaclass contains environement settings (Compiler, framework libraries, etc.) for WorkbenchManaged compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

AutoAttachToIDEDebugger

The property AutoAttachToIDEDebugger is used to specify that you would like the Workbench debugger to be automatically synchronized with the Rational Rhapsody animation. If for some reason you do not want automatic synchronization, for example, if you prefer to manually connect to the relevant target server, then you can set the value of this property to False.

Default = Checked

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to

True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=$Tool\" \"TOOL_FAMILY=$Tool\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Checked

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

Default = CXX = $(AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty MultiLine

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = vxmain

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(TOOL)$(LIB_EXT),
$(OMROOT)/lib/vxWebServices(CPU)(TOOL)$(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" Makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.

- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -X -r

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -X -r

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .makefile)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following

sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = OMROOT=$OMRoot INSTRUMENTATION=$OMInstrumentation LIBS+=$OMLibs LIB_EXT=.a TARGET_TYPE=$OMTargetType ConfigurationCPPCompileSwitches = $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ifeq ($(TOOL),diab) ConfigurationCPPCompileSwitches += $DiabCompileSwitches else ifeq ($(TOOL),gnu) ConfigurationCPPCompileSwitches += $GNUCompileSwitches endif endif INCLUDE_QUALIFIER=-I INCLUDE_PATH=$OMIncludePath ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(TOOL)$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(TOOL)$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(TOOL)$(LIB_EXT) SOCK_LIB= endif endif endif ADDED_INCLUDES+=$(INCLUDE_PATH) $(INST_INCLUDES) -I$AdaptorSearchPath -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf ADDED_C++FLAGS+=$(INST_FLAGS) $(ConfigurationCPPCompileSwitches) ifeq ($(TARGET_TYPE),Executable) ADDED_LIBS+=$(LIBS) $(OXF_LIBS) $(INST_LIBS) $(SOCK_LIB) SUB_OBJECTS+=$(LIBS) $(OXF_LIBS) $(INST_LIBS) $(SOCK_LIB) endif

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default =$(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated

specification (header) files for a given language and environment.

Default = .h

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

WorkbenchManaged_RTP

The WorkbenchManaged_RTP metaclass contains environement settings (Compiler, framework libraries, etc.) for WorkbenchManaged_RTP compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default = $(OMROOT)/LangCpp/osconfig/VxWorks

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

AutoAttachToIDEDebugger

The property AutoAttachToIDEDebugger is used to specify that you would like the Workbench debugger to be automatically synchronized with the Rational Rhapsody animation. If for some reason you do not want automatic synchronization, for example, if you prefer to manually connect to the relevant target server, then you can set the value of this property to False.

Default = Checked

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

Default = PENTIUM

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property CPP_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration.

To change this property, use the Configuration window in the browser - do not change it using the Properties tab in the Features window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag

(:cx_option=exceptions).

- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=$Tool\" \"TOOL_FAMILY=$Tool\" \"BUILD=$BuildCommandSet\" \"USE_RTP=TRUE\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Checked

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

Default = CXX = $(AMC_HOME)\bin\ctcxx

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty MultiLine

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The default is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CXX) $(C++FLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

Default = -g

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

Default = Empty string

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

Default = Checked

EnableDebugIntegrationWithIDE

When using Rational Rhapsody in conjunction with an IDE such as Eclipse, the property

EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

If applicable for the metaclass, see also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rational Rhapsody.

Note that the full name of the executable is composed of the value of the property CPP_CG::<Environment>::ExeName plus the value of this property.

(Default = .vxe)

ExeName

By default, the name of the executable created by Rational Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property CPP_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

Default = $OMSpecIncludeInElements $OMImpIncludeInElements

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default =

$(OMROOT)/LangCpp/lib/vxWebComponents(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT),
$(OMROOT)/lib/vxWebServices(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT)

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = Checked

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

Default = .cpp

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

Default = include

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" Makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = .a

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

Default = -X -r

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

Default = -X -r

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

Default = $OMLinkCommandSet

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property CPP_CG::<Environment>::MakeFileName.

If you do not want Rational Rhapsody to add a file extension, leave the value of this property blank.

(Default = .makefile)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

Default = OMROOT=$OMRoot INSTRUMENTATION=$OMInstrumentation LIBS+=$OMLibs RTP_SUFFIX = _RTP_ LIB_EXT=.a TARGET_TYPE=$OMTargetType ConfigurationCPPCompileSwitches = $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches ifeq ($(TOOL),diab) ConfigurationCPPCompileSwitches += $DiabCompileSwitches else ifeq ($(TOOL),gnu) ConfigurationCPPCompileSwitches += $GNUCompileSwitches endif endif INCLUDE_QUALIFIER=-I INCLUDE_PATH=$OMIncludePath ifeq ($(INSTRUMENTATION),Animation) INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS= $(OMROOT)/LangCpp/lib/vxaomanim(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),Tracing) INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom INST_LIBS=$(OMROOT)/LangCpp/lib/vxtomtrace(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxaomtrace(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) OXF_LIBS= $(OMROOT)/LangCpp/lib/vxoxfinst(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) $(OMROOT)/LangCpp/lib/vxomcomappl(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) SOCK_LIB= else ifeq ($(INSTRUMENTATION),None) INST_FLAGS= INST_INCLUDES= INST_LIBS= OXF_LIBS=$(OMROOT)/LangCpp/lib/vxoxf(CPU)(RTP_SUFFIX)$(TOOL)$(LIB_EXT) SOCK_LIB= endif endif endif ADDED_INCLUDES+=$(INCLUDE_PATH) $(INST_INCLUDES) -I$AdaptorSearchPath -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf ADDED_C++FLAGS+=$(INST_FLAGS) $(ConfigurationCPPCompileSwitches) ifeq ($(TARGET_TYPE),Executable) ADDED_LIBS+=$(LIBS) $(OXF_LIBS) $(INST_LIBS) $(SOCK_LIB) SUB_OBJECTS+=$(LIBS) $(OXF_LIBS) $(INST_LIBS) $(SOCK_LIB) endif

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rational Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property CPP_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default =$(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

Default = Empty string

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

Default = Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .o

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rational Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. The RemoteHost property can be left blank if both the application and Rational Rhapsody are running on the same machine.

Default = Empty string

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .h

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Checked

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rational Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rational Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Checked

UseTemplateTypename

In the declarations for members of a template class, some compilers require that you use precede the member type with the keyword "typename" for types with dependent names. Since support for this keyword varies between compilers, the property UseTemplateTypename is used to specify whether the "typename" keyword should be included in the generated code.

Default = Cleared

CPP_ReverseEngineering

In addition to the ReverseEngineering subject, Rational Rhapsody provides language-specific subjects to control how Rational Rhapsody imports legacy code. most of the properties are identical for each language. In general, most of the reverse engineering (RE) properties have graphical representation in the Reverse Engineering Options window. You should change the options using this window instead of the corresponding properties. The C++ metaclasses are as follows:

- ApproximatedConstructs
- Filtering
- ImplementationTrait
- Main
- MFC
- MSVC60
- Promotions

Filtering

The Filtering metaclass contains properties that control which items are analyzed during the reverse engineering operation.

AnalyzeGlobalFunctions

The AnalyzeGlobalFunctions property specifies whether to analyze global functions.

Default = Checked

AnalyzeGlobalTypes

The AnalyzeGlobalTypes property specifies whether to analyze global types.

Default = Checked

AnalyzeGlobalVariables

The AnalyzeGlobalVariables property specifies whether to analyze global variables.

Default = Checked

CreateReferenceClasses

The CreateReferenceClasses property specifies whether to create external classes for undefined classes that result fromforward declarations and inheritance.

By default, reference classes are created. If the incomplete class cannot be resolved, the tool deletes the incomplete class if this property is set to Cleared. In some cases, the class cannot be deleted (for example, a class referenced by a typedef type).

Default = Checked

IncludeInheritanceInReference

The IncludeInheritanceInReference property specifies whether to include inheritance information in reference classes.

Default = Cleared

ReferenceClasses

The ReferenceClasses property specifies which classes to model as reference classes. Reference classes are classes that can be mentioned in the final design as placeholders without having to specify their internal details. For example, you can include the MFC classes as reference classes, without having to specify any of their members or relations.

They would simply be modeled as terminals for context, to show that they are acting as superclasses or peers to other classes.

Default = empty string

ReferenceDirectories

The ReferenceDirectories property specifies which directories (and subdirectories) contain reference classes.

Default = empty string

ImplementationTrait

The ImplementationTrait metaclass contains properties that determine the implementation traits used during the reverse engineering operation.

AnalyzeIncludeFiles

The AnalyzeIncludeFiles property specifies which, if any, include files should be analyzed during reverse engineering. The possible values are as follows:

- AllIncludes - Analyze all include files.
- IgnoreIncludes - Ignore all include files.
- OnlyFromSelected - Analyze the specified include files only.
- OnlyLogicalHeader - Analyze the logical header files only.

Default = OnlyFromSelected

AutomaticIncludePath

When Rational Rhapsody reverse engineers a file, there may be cases where the file references a header file but the path in the include directive is not clear enough for Rational Rhapsody to find the file. If you set the value of the property AutomaticIncludePath to Checked, then in such cases, Rational Rhapsody searches the list of files to be reverse engineered to see if the list contains a header file with that name. If there is such a file, Rational Rhapsody uses the full path that was provided for that header file, assuming that this is the header file that was being referenced in the original file.

Rational Rhapsody performs this search for ambiguous header files when it does macro collection. This means that if the value of the property CPP_ReverseEngineering::ImplementationTrait::CollectMode is set to None, then Rational Rhapsody does not search for ambiguous header files even if the value of the property AutomaticIncludePath is set to Checked.

Default = Checked

CreateBlackDiamondAssociations

The property CreateBlackDiamondAssociations specifies how the reverse engineering feature should handle composition relationships. If the value of the property is set to False, then Rational Rhapsody creates parts. If the value of the property is set to Checked, Rational Rhapsody creates composition associations (black diamond).

Default = Cleared

CollectMode

The CollectMode property allows Rational Rhapsody to collect macros. The possible values are as follows:

- None - Macros are not collected from include files that are not on the reverse engineering list.
- Once - Macros are collected only if the model does not yet include a controlled file of collected macros.
- Always - Macros are collected each time reverse engineering is carried out. The controlled file that stores the macros are replaced each time.

Default = Once

ComponentFileType

The possible values are "SpecificationOrImplementation" or "Logical."

Default = SpecificationOrImplementation

CreateDependencies

The property CreateDependencies allows you to specify how the Reverse Engineering feature should handle the creation of dependency elements in the model from code constructs such as #includes, forward declarations, friends, and namespace usage.

The default value for this property represents the most advanced handling option available in Rational Rhapsody, while some of the other options represent various handling options that were introduced in earlier versions of Rational Rhapsody.

The possible values for this property are:

- None - Dependencies are not created to represent code constructs such as #includes and forward declarations.
- DependenciesOnly - Dependencies are created in the model only when the code represents a dependency relationship between two classes.
- PackageOnly - Actual dependencies are created in the model only when the code represents a dependency relationship between two classes. For other #include statements, the relevant information is stored in the properties SpecIncludes and ImpIncludes so that the original code can be regenerated.
- ComponentOnly - Dependencies are created for all code constructs that represent dependency relationships, however the dependency is created between the component files that contain the elements rather than between the elements themselves.
- PackageAndComponent - Dependencies are created for all code constructs that represent dependency relationships. For each such relationship, a dependency is created between the component files containing the relevant elements, and where possible a dependency is also created between the relevant elements themselves. This means that for some #includes two dependencies are created in the model.
- SmartPackageAndComponent - Dependencies are created for all code constructs that represent dependency relationships. Where possible, the dependency is created between the relevant elements. Where this is not possible, a dependency is created between the component files containing the elements. This means that only a single dependency is created for any single #include statement.

Default = SmartPackageAndComponent

CreateFilesIn

The CreateFilesIn property is a placeholder for the reverse engineering option Create File-s In option. You should not set this value directly. The C++ default value is None.

DataTypesLibrary

The Mapping tab of the Reverse Engineering Options dialog allows you to specify a list of types that should be modeled as "Language" types. You can add individual types to the list or groups of types that you have previously defined as data types for a specific library.

If you select the option of adding a library, you are presented with a drop-down list of libraries to choose from. The libraries on this list are taken from the value of the property DataTypesLibrary. You can add a number of libraries to the drop-down list by using a comma-separated list of names as the value for this property.

When you select a library from the drop-down list, all of the types that were defined for that library are added to the list of types.

You define types for a library by carrying out the following steps:

- In the relevant .prp file, under the subject [lang]_ReverseEngineering, add a metaclass with the name of the library (using the same name you used in the value of the property DataTypesLibrary).
- Under the new metaclass, add a property called DataTypes.
- For the value of the DataTypes property that you added, enter a comma-separated list of the types that you want to include for that library.
- Now, if you select the library from the drop-down list displayed on the Mapping tab, the types you defined with the DataTypes property is automatically added to the list of types that should be modeled as "Language" types.

Default = MFC

ImportAsExternal

The property ImportAsExternal specifies whether the elements contained in the files you are reverse engineering should be brought into the model as "external" elements. This means that code will not be generated for these elements during code generation.

This property corresponds to the Import as External check box on the Mapping tab of the Reverse Engineering Options dialog.

Default = Cleared

ImportDefineAsType

The ImportDefineAsType property is a Boolean value that specifies how to import a #define. Note that models created before Version 5.2 automatically have this property overridden (set to True) when the model is loaded. The possible values are as follows:

- True - Import a #define as a user type.
- False - Import a #define as a constant variable, constant function, or type according to the following policy:
- If the #define has parameters, Rational Rhapsody creates a constant function. This applies to Rational Rhapsody Developer for C only.
- If the #define does not have parameters and its value includes only one line, Rational Rhapsody creates a constant variable. In Rational Rhapsody Developer for C++, the property CG::Attribute::ConstantVariableAsDefine is set to True.
- If the #define was not imported as a variable or function, Rational Rhapsody creates a type.

Default = False

ImportGlobalAsPrivate

The ImportGlobalAsPrivate property allows you to import C functions as public or private.

The possible values are as follows:

- Never - Import globals (functions) as public. The declaration remains in the specification file.
- InImplementation - Global functions are imported as private. Both the declaration and the implementation of the function are imported into the implementation (.c) file.
- StaticInImplementation - Globals are imported as private in the implementation (.c) file and the functions are marked as static. (same as "InImplementation" but the keyword "static" is added to the declaration and implementation of the function).

ImportStructAsClass

The ImportStructAsClass property is a Boolean value specifies how structs in external code are imported during reverse engineering. The possible values are as follows:

- Checked - structs are imported as classes.
- Cleared - structs are imported as types of kind Structure.

Default = Cleared

LocalizeRespectInformation

When reverse engineering code in Respect mode, Rational Rhapsody stores information such as the order of code elements so that when code is regenerated from the model, the code will resemble as much as possible the original code.

When the property LocalizeRespectInformation is set to Checked, the software stores this information as SourceArtifact elements below the relevant class. (These elements are not visible by default, but you can see them in the model if you set the value of the property ShowSourceArtifacts to True.)

If the value of the property LocalizeRespectInformation is set to Cleared, then Rational Rhapsody stores this "respect" information as File elements under the relevant Component.

Default = Checked

MacroExpansion

Early versions of Rational Rhapsody were not capable of importing macros in code such that they would be regenerated as macros. Rather, the code represented by the macro was stored in the model, and when the code was regenerated, the macro calls would be replaced with the relevant code.

Now, by default, Rational Rhapsody imports macros such that when the code is regenerated, the macro

definition and macro calls are generated as they appeared in the original code that was reverse engineered.

If you would like the previous Rational Rhapsody behavior, i.e., replacement of macro calls with the actual macro code, you can set the property MacroExpansion to Checked.

Note that the property CPP_ReverseEngineering::Parser::ForceExpansionMacros allows you to specify that individual macros should be expanded during reverse engineering even if the value of the property MacroExpansion is set to False.

Default = Cleared

MapGlobalsToComponentFiles

The property MapGlobalsToComponentFiles allows you to specify whether Rational Rhapsody should map global variables, functions, and types to component files, reflecting the original file location of these elements in the files that were reverse engineered. The property can take any of the following values:

- True - All global variables, functions, and types should be mapped to component files
- OnExternal - Global variables, functions, and types should be mapped to component files only if the user selected the reverse engineering option "Import as External"
- TypesOnly - Global types should be mapped to component files, but not global variables and functions
- TypesOnExternal - Only global types should be mapped to component files, and this should only be done if the user selected the reverse engineering option "Import as External"
- False - Global variables, functions, and types should not be mapped to component files

Default = True

MapToPackage

The property MapToPackage allows you to specify how the code elements you are reverse engineering should be divided into packages.

The property represents the options that appear in the Map to Package section of the Mapping tab in the Reverse Engineering Options dialog.

When the value of the property is set to Directory, a separate package is created for each subdirectory in the directory you have chosen to reverse engineer. The elements found in the files in each subdirectory is added to the package that corresponds to that subdirectory.

If you set the value of this property to User, then Rational Rhapsody places all of the reverse engineered elements into a single package in the model. The name of the package is taken from the property [lang]_ReverseEngineering::ImplementationTrait::UserPackage.

Default = Directory

ModelStyle

The property ModelStyle determines how model elements are opened in the browser after reverse engineering - using a file-based functional approach or using an object-oriented approach based on classes (the corresponding property values are Functional and ObjectBased).

This property corresponds to the Modeling Policy radio buttons on the Mapping tab of the Reverse Engineering Options window.

Note that for C++ and Java, the file-based approach can only be used for visualization purposes. Rational Rhapsody does not generate code from the model for elements imported using the Functional option. (You will notice that in the Reverse Engineering Options window, you can only select the File radio button if you first select the Visualization Only option.)

Default = Functional in RiC, ObjectBased in RiC++ and RiJ

PackageForExternals

If the value of the property UsePackageForExternals is set to True, the Rational Rhapsody reverse engineering feature puts all external elements in a separate package. You can control the name of this package by changing the value of the property PackageForExternals.

Default = Externals

PreCommentSensibility

During reverse engineering, a comment that comes immediately before the code for an element is considered a comment for that element, and the comment text is brought into Rational Rhapsody as the description for that element.

The property PreCommentSensibility is used to specify the maximum number of lines by which a comment can precede the code for an element and still be considered a comment for that element. Any comment that precedes an element by more than the number of lines specified is considered a global comment.

A value of 1 means that a comment must appear on the line prior to the code for an element to be considered a comment for that element.

Default = 2

ReflectDataMembers

The property ReflectDataMembers determines how the visibility of attributes is brought into the model when code is reverse engineered. The property affects both the visibility of the attribute in the regenerated code and the generation of get and set operations for the attribute. The property can take any of the following values:

- None - The visibility used for attributes is the same as that specified in the code that was reverse engineered. However, Rational Rhapsody generates public get/set operations for the attributes regardless of the visibility specified.
- VisibilityOnly - The visibility used for attributes is the same as that specified in the code that was

reverse engineered. In addition, Rational Rhapsody generates get/set operations for the attribute with the same visibility. For example, if the attribute visibility in the original code was private, the visibility is private in the regenerated code and the code will also include private get/set operations for the attribute.

- VisibilityAndHelpers - The visibility used for attributes is the same as that specified in the code that was reverse engineered. Rational Rhapsody does not generate get/set operations for the attribute if the original code did not contain such operations.

Note that when the property is set to VisibilityAndHelpers, not only will get/set operations not be generated for attributes, but Rational Rhapsody does not generate any of its automatically-generated operations such as default constructors.

Default = VisibilityAndHelpers

RespectCodeLayout

The property RespectCodeLayout determines to what degree Rational Rhapsody attempts to save information about the code that is reverse engineered so that it is possible to match the original code when code is later regenerated from the model. This includes things like:

- order of #includes and other code elements
- handling of preprocessor directives such as #ifdefs
- keeping macro calls as they were rather than expanding the macro in the regenerated code
- handling of global comments

The property can take any of the following values:

- None - Rational Rhapsody does not save information about the order of elements in the code that is imported, nor does it save the information necessary to regenerate all elements back to the files from which they were originally imported.
- Mapping - Rational Rhapsody saves the partial information so that it can regenerate all elements back to the files from which they were originally imported.
- Ordering - Rational Rhapsody saves all of the information it can so that the regenerated code matches the original code as much as possible. See the examples listed above.

Note that even if the value of this property is set to Ordering, Rational Rhapsody only attempts to match the regenerated code to the original code if the property [lang]_CG::Configuration::CodeGeneratorTool is set to Advanced, which is the default value for that property.

Default = Ordering

RootDirectory

This property specifies the root directory for reverse engineering. This root directory may contain all the folders that should become package during the reverse engineering process. Rational Rhapsody builds the package hierarchy according to the folder tree from the specified path.

Default = empty string

UseCalculatedRootDirectory

This property controls the use of the <lang>_ReverseEngineering::Implementation::RootDirectory property.

The possible values are:

- Never - Do not calculate the root directory.
- Always - Calculate the root directory and override the RootDirectory property.
- Auto - Ask the user if they want to override the value in the RootDirectory property if it is different from the calculated root directory. If the RootDirectory property is empty, Rational Rhapsody uses the calculated value without asking.

Default = Auto

UsePackageForExternals

When Rational Rhapsody generates code, it does not regenerate code for elements that have been brought in as "external" elements. By default, the reverse engineering feature puts all external elements in a separate package in the model. You can change this behavior by changing the value of the property UsePackageForExternals. When a separate package is used, the name of the package is taken from the value of the property PackageForExternals.

Default = Checked

UserDataTypes

The UserDataTypes specifies classes to be modeled as data types. This property corresponds to types entered in the Add Type window.

Default = empty string

UserPackage

When reverse engineering files, Rational Rhapsody allows you the option of having packages created for each subdirectory or having all of the reverse-engineered elements placed in a single package. This option is controlled by the property [lang]_ReverseEngineering::ImplementationTrait::MapToPackage.

When MapToPackage is set to "User", you can use the property UserPackage to provide the name that you would like Rational Rhapsody to use for the single package that will contain all of the reverse-engineered elements.

You can specify a nested package by using the following syntax: package1::package2

If the model already contains a package with the specified name, the reverse-engineered elements are put in that package. If not, Rational Rhapsody creates the package.

This property corresponds to the text field provided for the package name in the Map to Package section of the Mapping tab in the Reverse Engineering Options dialog.

Default = ReverseEngineering

Main

The metaclass Main contains properties that define the file extensions used for filtering files in the reverse engineering file selection dialog, as well as properties that enable jumping to problematic lines of code by double-clicking messages in the Output window.

ErrorMessageTokensFormat

When errors are encountered during reverse engineering, they are displayed in the Rational Rhapsody Output window. If you double-click the error message, you are taken to the problematic line in the relevant source file.

This ability is made possible by the values provided for the properties ParseErrorMessage and ErrorMessageTokensFormat.

The value of the property ParseErrorMessage is a regular expression that extracts the relevant filename and line number information from the Rational Rhapsody-generated error message. The value of the property ErrorMessageTokensFormat is then used to interpret the information that was extracted from the error message.

The value of the property ErrorMessageTokensFormat consists of a comma-separated list of keyword-value pairs representing the number of tokens contained in the extracted information, which token represents the filename, and which token represents the line number.

Users should not change the value of this property.

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ImplementationExtension

The ImplementationExtension property specifies the file extensions used to filter the list of files displayed in the Add Files window of the reverse engineering tool.

The C++ default values are c,cpp,cxx,cc.

ParseErrorMessage

When errors are encountered during reverse engineering, they are displayed in the Rational Rhapsody Output window. If you double-click the error message, you are taken to the problematic line in the relevant source file.

This ability is made possible by the values provided for the properties ParseErrorMessage and ErrorMessageTokensFormat.

The value of the property ParseErrorMessage is a regular expression that extracts the relevant filename and line number information from the Rational Rhapsody-generated error message. The value of the property ErrorMessageTokensFormat is then used to interpret the information that was extracted from the error message.

Users should not change the value of this property.

Default = "([a-zA-Z_]+[:0-9a-zA-Z_.\/])"[:][ ]*LINE[ ]*([0-9]+)*

SpecificationExtension

The property SpecificationExtension is used to specify the filename extensions that should be used to filter files in the reverse engineering file selection dialog. This property is used in conjunction with the property ImplementationExtension.

You can specify a number of extensions. They should be entered as a comma-separated list.

Default = h,hpp,hxx,inl

MFC

The MFC metaclass contains a property that affects the MFC type library.

DataTypes

The DataTypes property specifies classes to be modeled as MFC data types. There is only one predefined library (MFC) that contains only one class (Cstring). You can, however, expand this short list of classes by the addition of classes in this property or the creation of new libraries in the property files factory.prpfactory and site.prpsite.

Default = Cstring

MSVC60

The MSVC60 metaclass contains properties used to control the Microsoft Visual C++ environment.

Defined

The Defined property specifies symbols that are defined for the Microsoft Visual C++ version 6.0

(MSVC60) preprocessor. These symbols are automatically filled into the Name list of the Preprocessing tab of the Reverse Engineering Options window when you select Add > Dialect: MSVC60.

The default value is as follows: __STDC__,__STDC_VERSION__,__cplusplus,__DATE__, __TIME__,_WIN32,_cdecl,__cdecl,__int64=int,__stdcall, __export,_export,_AFX_PORTABLE,_M_IX86=500,__declspec, __MSC_VER=1200,__inline=inline,__far,__near,_far,_near, __pascal,_pascal,__asm,__finally=catch,__based, __inline=inline,__single_inheritance,__cdecl,__int8=int, __stdcall,__declspec,__int16=int,__int32=int,__try=try, __int64=int,__virtual_inheritance,__except=catch, __leave=catch,__fastcall,__multiple_inheritance)

IncludePath

The IncludePath property specifies necessary include paths for the Microsoft Visual C++ preprocessor. It is possible to specify the path to the site installation of the compiler as part of the site.prp, thus doing it only once and not for every project.

Default = empty string

Undefined

The Undefined property specifies symbols that must be undefined for the Microsoft Visual C++ preprocessor.

Default = empty string

Parser

The metaclass Parser contains properties that can be used to modify the way the parser handles code during reverse engineering.

AdditionalKeywords

The property AdditionalKeywords can be used to list non-standard keywords that may appear in the code that you reverse engineer. This allows Rational Rhapsody to parse this code correctly during reverse engineering.

The value of this property should be a comma-separated list of the additional keywords you want to include.

Note that keywords with parameters are not supported, nor are keywords that consist of more than one word.

This property corresponds to the keywords listed on the Preprocessing tab of the Reverse Engineering

Options window. Note that when you add additional keywords using the controls on the Preprocessing tab, these keywords are included in the value of the AdditionalKeywords property at the level of the active configuration.

Default = far,near

Defined

The Defined property specifies symbols and macros to be defined using #define. For example, you can enter the following to define name> as text with the appropriate intermediate character: /D name{=|#}text

Default = empty string

Dialects

The Dialects property specifies which symbols are added to the Preprocessing tab of the Reverse Engineering dialog box when that dialect is selected. The default value is MSVC60, which is itself defined by a metaclass of the same name under subject CPP_ReverseEngineering. This dialect specifies the symbols that are defined for the Microsoft Visual C++ environment. You can define your own dialect (in the site.prp file) and select it in the Dialects property. The default value for C is an empty string; the default value for C++ is MSVC60.

ForceExpansionMacros

By default, Rational Rhapsody reverse engineers macros such that when the code is regenerated, the macro definition and macro calls are generated as they appeared in the original code that was reverse engineered. (This behavior can be controlled with the property CPP_ReverseEngineering::ImplementationTrait::MacroExpansion.)

In some cases, you may find that you are not satisfied with the way that Rational Rhapsody imports the macro. For such situations, you can use the property ForceExpansionMacros to list specific macros that should be expanded during reverse engineering even if the value of the property MacroExpansion is set to False.

The value of this property should be a comma-separated list of the macros that you would like Rational Rhapsody to expand during reverse engineering.

Default = Blank

IncludePath

The Preprocessing tab of the Reverse Engineering Options dialog allows you to specify an include path (classpath for Java) for the parser to use. The property IncludePath represents this path.

For the value of this property, you can enter a comma-separated list of directories. Note that you have to specify subdirectories individually.

The directories you list here is combined with the directories specified in #include statements in order to

find the necessary files. For example, if you have c:\d1\d2\d3\file.h, you can enter c:\d1\d2 as the value of this property and then use d3\file.h in the #include statement.

You should take into account that the value of this property also determines the structure of the source file directory when code is generated from the model. So, in the above example, the top-level directory created is d3.

Default = Blank

Undefined

The Undefined property specifies symbols and macros to be undefined using #undef.

Default = empty string

Promotions

The metaclass Promotion contains a number of properties used to specify whether Rational Rhapsody should add various advanced modeling constructs to your model based on relationships/patterns uncovered during reverse engineering.

EnableAttributeToRelation

The property EnableAttributeToRelation is used to specify whether Rational Rhapsody should add Associations to the model for attributes whose type is another class in the model.

For example, if you have two classes, A and B, and B contains an attribute of type A, Rational Rhapsody adds an Association to the model reflecting this relationship.

Default = Checked

EnableFunctionToObjectBasedOperation

The EnableFunctionToObjectBasedOperation property specifies whether object-based promotion is enabled during reverse engineering. Object-based promotion “promotes” a global function to comply with the pattern specified in the properties C_CG::Operation::PublicName and ProtectedName to be an operation of the class (object_type) defined in the function’s me parameter.

Default = Cleared

EnableResolveIncompleteClasses

Sometimes, during reverse engineering, Rational Rhapsody is not able to find the base class for a given class. The property EnableResolveIncompleteClasses is used to specify that if Rational Rhapsody finds a

class with the same name as the base class in a different location, it should assume that this class is the missing base class.

Default = Checked

EnableTypeToTemplateInstantiation

During reverse engineering, when Rational Rhapsody encounters a typedef declaration that contains a template instantiation, it will by default create a template instantiation in the model.

The property EnableTypeToTemplateInstantiation allows you to change this behavior. If you set the value of this property to False, then during reverse engineering Rational Rhapsody creates a Language type in the model rather than a template instantiation.

Default = Checked

Update

The metaclass Update contains properties used to control various aspects of the Rational Rhapsody behavior during and after reverse engineering.

CreateFlowcharts

The property CreateFlowcharts is used to specify whether or not Rational Rhapsody should automatically create flowcharts for operations during reverse engineering of code.

This property corresponds to the Create Flowcharts option on the Model Updating tab of the Reverse Engineering options window. Note that when you select the Create Flowcharts option, the value of the property CreateFlowcharts is modified at the level of the active configuration.

This property can be used in conjunction with the properties FlowchartCreationCriterion, FlowchartMinLOC, FlowchartMaxLOC, FlowchartMinControlStructures and FlowchartMaxControlStructures so that flowcharts are created only for operations that are within a given range in terms of lines of code or in terms of the number of control structures in the operation.

Default = Cleared

FlowchartCreationCriterion

If you have selected the option of having Rational Rhapsody create flowcharts during reverse engineering, you can use the property FlowchartCreationCriterion to select the criterion that should be used to decide what operations Rational Rhapsody should create flowcharts for.

The property corresponds to the radio buttons on the Model Updating tab of the Reverse Engineering options window.

The property can take the following values:

- Control Structures - the decision whether or not to generate a flowchart for an operation is based on the number of control structures in the operation. When this option is selected, the minimum and maximum number of control structures used to define the inclusion criterion are taken from the properties FlowchartMinControlStructures and FlowchartMaxControlStructures.
- LOC - the decision whether or not to generate a flowchart for an operation is based on the number of lines of code in the operation. When this option is selected, the minimum and maximum lines of code used to define the inclusion criterion are taken from the properties FlowchartMinLOC and FlowchartMaxLOC.

Default = LOC

FlowchartMaxControlStructures

If you have selected the option of having Rational Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to Control Structures, then the property FlowchartMaxControlStructures is used to specify the maximum number of control structures that an operation can have, above which Rational Rhapsody does not create a flowchart for it.

The property corresponds to the maximum control structures text box on the Model Updating tab of the Reverse Engineering options window.

Default = 10

FlowchartMaxLOC

If you have selected the option of having Rational Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to LOC, then the property FlowchartMaxLOC is used to specify the maximum number of lines of code that an operation can have, above which Rational Rhapsody does not create a flowchart for it.

The property corresponds to the maximum lines of code text box on the Model Updating tab of the Reverse Engineering options window.

Default = 100

FlowchartMinControlStructures

If you have selected the option of having Rational Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to Control Structures, then the property FlowchartMinControlStructures is used to specify the minimum number of control structures that an operation must have in order to have Rational Rhapsody create a flowchart for it.

The property corresponds to the minimum control structures text box on the Model Updating tab of the Reverse Engineering options window.

Default = 2

FlowchartMinLOC

If you have selected the option of having Rational Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to LOC, then the property FlowchartMinLOC is used to specify the minimum number of lines of code that an operation must have in order to have Rational Rhapsody create a flowchart for it.

The property corresponds to the minimum lines of code text box on the Model Updating tab of the Reverse Engineering options window.

Default = 10

CPP_Roundtrip

The CPP_Roundtrip subject contains properties that affect roundtripping.

The metaclasses are as follows:

- General
- Update

General

The General metaclass contains properties that control how changes to code are roundtripped in Rational Rhapsody.

CreateFileAsUnit

When a File is created in a model, the value of the property General::Model::FileIsSavedUnit determines whether or not it is saved as a unit (i.e., as a separate file in the file system).

The property CreateFileAsUnit provides you with a certain degree of flexibility during roundtripping so that a File created during roundtripping can be saved as a unit even if the value of the property FileIsSavedUnit is set to False.

The possible values for this property are:

- Default - The decision whether or not to save the newly-created File as a unit is based on the value of the property FileIsSavedUnit.
- AsModel - The decision whether or not to save the newly-created File as a unit will depend on the class in the model that it represents. If the class is currently saved as a unit, then the File created during roundtripping will also be saved as a unit. If the class is not a unit, then the File created will also not be saved as a unit.

Default = AsModel

CreateFolderAsUnit

When a Folder is created in a model, the value of the property General::Model::FolderIsSavedUnit determines whether or not it is saved as a unit (i.e., as a separate file in the file system).

The property CreateFolderAsUnit provides you with a certain degree of flexibility during roundtripping so that a Folder created during roundtripping can be saved as a unit even if the value of the property FolderIsSavedUnit is set to False.

The possible values for this property are:

- Default - The decision whether or not to save the newly-created Folder as a unit is based on the value

of the property FolderIsSavedUnit.

- AsModel - The decision whether or not to save the newly-created Folder as a unit will depend on the package in the model that it represents. If the package is currently saved as a unit, then the Folder created during roundtripping will also be saved as a unit. If the package is not a unit, then the Folder created will also not be saved as a unit.

Default = AsModel

NotifyOnInvalidatedModel

The NotifyOnInvalidatedModel property is a Boolean value that determines whether a warning window is displayed during roundtrip. This warning is displayed when information might get lost because the model was changed between the last code generation and the roundtrip operation.

(Default = Checked)

ParserErrors

The ParserErrors property specifies the behavior of roundtrip when a parser error is encountered. The possible values are as follows:

- Abort—Abort roundtrip whenever there is a parser error in the code. No changes is applied to the model.
- AskUser—When Rational Rhapsody encounters an error, it asks what you want to do.
- AbortOnCritical—Abort roundtrip if any critical parser errors are encountered in the code.
- Ignore—Continue roundtrip, ignoring any parser errors that are encountered.

(C++ Default = AskUser)

PredefineIncludes

The PredefineIncludes property specifies the predefined include path for roundtripping.

The default value for C++ is an empty string.

PredefineMacros

The PredefineMacros property specifies the predefined macros for roundtripping. The default value is as follows:

DECLARE_META(class_0\,animClass_0), DECLARE_REACTIVE_META(class_0\,animClass_0),
OMINIT_SUPERCLASS(class_0Super\,animClass_0Super),
OMREGISTER_CLASS\,DECLARE_META_T(class_0\, ttype\,animClass_0),
DECLARE_REACTIVE_META_T(class_0\, ttype\,animClass_0),
DECLARE_META_SUBCLASS_T(class_0\, ttype\,animClass_0),
DECLARE_REACTIVE_META_SUBCLASS_T(class_0\, ttype\,animClass_0),
DECLARE_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM),

IMPLEMENT_META(class_0\,Default\,FALSE),
IMPLEMENT_META_S(class_0\,FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_META_M(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_REACTIVE_META(class_0\,Default\,FALSE),
IMPLEMENT_REACTIVE_META_S(class_0\,FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_REACTIVE_META_SIMPLE(class_0\,Default\,FALSE),
IMPLEMENT_REACTIVE_META_S_SIMPLE(class_0\,FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M_SIMPLE(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_META_T(class_0\, Default\, FALSE\, animClass_0),
IMPLEMENT_META_S_T(class_0\,FALSE\,class_0Super\,animclass_0Super\,animClass_0),
IMPLEMENT_META_M_T(class_0\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_META_T_S_T(tname\,IsSingleton\,SuperClass\,animSuperClass\,animTname),
IMPLEMENT_META_T_S_T_N(tname\,IsSingleton\,NameSpace\,SuperClass\,animSuperClass\,animTname),
IMPLEMENT_META_OBJECT(class_0\,class_type\,Default\,FALSE),
IMPLEMENT_META_S_OBJECT(class_0\,class_type\,FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_META_M_OBJECT(class_0\,class_type\,FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_REACTIVE_META_OBJECT(class_0\,class_type\,Default\,FALSE),
IMPLEMENT_REACTIVE_META_S_OBJECT(class_0\,class_type\,FALSE\,class_1\,animClass_1\,animClass_0),
IMPLEMENT_REACTIVE_META_M_OBJECT(class_0\,class_type\,FALSE\, class_0Super\, 2
\,animClass_0),
IMPLEMENT_REACTIVE_META_SIMPLE_OBJECT(class_0\,class_type\,Default\,FALSE),
IMPLEMENT_REACTIVE_META_S_SIMPLE_OBJECT(class_0\,class_type\,FALSE\,class_1\,animClass_1\,animClass_0
IMPLEMENT_REACTIVE_META_M_SIMPLE_OBJECT(class_0\,class_type\,FALSE\, class_0Super\,
2 \,animClass_0), IMPLEMENT_META_T_OBJECT(class_0\,class_type\, Default\, FALSE\,
animClass_0),
IMPLEMENT_META_S_T_OBJECT(class_0\,class_type\,FALSE\,class_0Super\,animclass_0Super\,animClass_0),
IMPLEMENT_META_M_T_OBJECT(class_0\,class_type\, FALSE\, class_0Super\, 2 \,animClass_0),
IMPLEMENT_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM\,INCREMENTNUM\,ISPROTECTED),
DECLARE_META_PACKAGE(Default), DECLARE_PACKAGE(Default),
IMPLEMENT_META_PACKAGE(Default\,Default), DECLARE_META_EVENT(event_0),
DECLARE_META_SUBEVENT(event_0\,event_0Super\,event_0SuperNamespace),
IMPLEMENT_META_EVENT(event_0\,Default\,event_0), IMPLEMENT_META_EVENT_S(words\,
words\, baseWords), DECLARE_OPERATION_CLASS(mangledName),
DECLARE_META_OP(mangledName), OM_OP_UNSER(type\, name), OP_UNSER(func\, name),
OP_SET_RET_VAL(retVal), OM_OP_SET_RET_VAL(retVal),
IMPLEMENT_META_OP(animatedClassName\, mangledName\, opNameStr\, isStatic\, signatureStr\,
numOfArgs), IMPLEMENT_OP_CALL(mangledName\, userClassName\, call\, retExp),
STATIC_IMPLEMENT_OP_CALL(mangledName\, userClassName\, call\, retExp),
OMDefaultThread=0, NULL=0, OMDECLARE_GUARDED, OM_DECLARE_COMPOSITE_OFFSET

ReportChanges

The ReportChanges property defines which changes are reported (and displayed) by the roundtrip operation. The possible values are as follows:

- None—No changes are displayed in the output window.
- AddRemove—Only the elements added to, or removed from, the model are displayed in the output window.
- UpdateFailures—Only unsuccessful changes to the model are displayed in the output window.
- All—All changes to the model are displayed in the output window.

(Default = AddRemove)

RestrictedMode

The RestrictedMode property is a Boolean value that specifies whether restricted-mode roundtripping is enabled. This property can be modified on the configuration level.

Restricted mode of full roundtrip enables you to roundtrip unusual usage of Rational Rhapsody elements, such as a class declaration in a user-defined type. Restricted mode has more limitations, but preserves the model from unexpected changes. The additional limitations for restricted mode are as follows:

- User-defined types cannot be removed or changed on roundtrip because Rational Rhapsody code generation adds the Ignore annotation for a user-defined type declaration.
- Relations cannot be removed or changed on roundtrip.
- New classes are not added to the model.

(Default = Cleared)

RoundtripPreprocessorDirectives

By default, the Rational Rhapsody roundtripping feature takes into account changes made to preprocessor directives. The property RoundtripPreprocessorDirectives can be used to turn off roundtripping for the following types of preprocessor directives:

- elif
- else
- endif
- error
- if
- ifdef
- ifndef
- import
- line
- pragma
- undef
- using

Default = Checked

RoundtripScheme

Determines what type of changes can be roundtripped back into the model. The possible values are Basic, Advanced, and Respect.

When set to Basic, only changes to the bodies of operations and actions are roundtripped into the model.

When set to Advanced, roundtripping also takes into account elements that have been added, such as attributes and operations, and can optionally take into account elements that have been modified or removed.

When set to Respect, roundtripping also takes into account the changes that are covered by the Rational Rhapsody "code respect" feature, for example, the order of class members or elements like #include-s.

(Note that for pre-7.1 models, the possible values are Basic and Full, where Basic roundtrips only changes to the bodies of operations and actions, and Full represents the roundtripping mode currently referred to as Advanced.)

Default = Respect

Update

The Update metaclass contains a property that controls the update process used during roundtripping.

AcceptChanges

The AcceptChanges property is an enumerated type that specifies which changes are applied to each CG element (attribute, operation, type, class, or package). You can apply separate properties to each type of CG element. The possible values are as follows:

- All—All the changes can be applied to the model element, including deletion.
- Default—1) Rational Rhapsody does not roundtrip deletions if the updated code results in parser errors. 2) Rational Rhapsody does not roundtrip the deletion of classes.
- NoDelete—All the changes except deletion can be applied to the model element. This setting prevents accidental removal of operations, constructors, attributes, relations, variables, instances, and functions.
- AddOnly—Apply only the addition of an aggregate to the model element. You cannot delete or change elements.
- NoChanges—Do not apply any changes to the model element.

Note that the value of the property is propagated to all the aggregates of an element. Therefore, if a package has the property value NoChanges, no elements in that package is changed.

Default = "Default" (in code-centric settings, default value is All)

C_CG

The C_CG subject contains several metaclasses for operating system environments and the following general metaclasses:

- Argument
- Attribute
- Class
- Configuration
- Cygwin
- Dependency
- Event
- File
- Framework
- General
- INTEGRITY
- INTEGRITY5
- Link
- Linux
- Microsoft
- MicrosoftIDF
- ModelElement
- Multi4Win32
- NucleusPLUS-PPC
- Operation
- Package
- Port
- Relation
- Solaris2
- Solaris2GNU
- Statechart
- Type
- VxWorks
- VxWorks6diab
- VxWorks6diab_RTP
- VxWorks6gnu
- VxWorks6gnu_RTP
- WorkbenchManaged

- WorkbenchManaged_RTP

Argument

The Argument metaclass contains properties that control how arguments are generated in code.

DeclarationModifier

The property DeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in argument declarations. Keywords that appear between the argument type and the argument name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and PostDeclarationModifier.

Default = Blank

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified element’s tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords

- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

IsRegister

The property IsRegister can be used to specify that the keyword "register" should be generated in the code for a given argument.

Default = Cleared

IsVolatile

The property IsVolatile allows you to specify that a specific operation argument should be declared as volatile.

Default = Cleared

PostDeclarationModifier

The property PostDeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in argument declarations. Keywords that appear after the argument name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and DeclarationModifier.

Default = Blank

PreDeclarationModifier

The property PreDeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in argument declarations. Keywords that appear before the argument type are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties DeclarationModifier and PostDeclarationModifier.

Default = Blank

Attribute

The Attribute metaclass contains properties that control attributes of code generation, such as whether to generate accessor operations.

AccessorGenerate

The AccessorGenerate property specifies whether to generate accessor operations for attributes. The possible values are as follows:

- Checked - A get() method is generated for the attribute.
- Cleared - A get() method is not generated for the attribute. This is the default value for C.

Setting this property to Cleared is one way to optimize your code for size.

AccessorVisibility

The AccessorVisibility property specifies the access level of the generated accessor for attributes. This enables you to define the access level of an accessor operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute’s access level for the accessor.
- public - Set the accessor’s access level to public.
- private - Set the accessor’s access level to private.
- protected - Set the accessor’s access level to protected. This value is not available in Rational Rhapsody Developer for C.

(Default = fromAttribute)

AttributeInitializationFile

The AttributeInitializationFile property specifies how static const attributes are initialized. In Rational Rhapsody, you can initialize these attributes in the specification file or directly in the initialization file. This property is analogous to the VariableInitializationFile property for global const variables. The possible values are as follows:

- Default - The attribute is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize constant attributes in the implementation file.
- Specification - Initialize constant attributes in the specification file.

(Default = Default)

BitField

Allows you to define a bit field for an attribute. To define a bit field, open the Features dialog for the relevant attribute and enter the number you want to use for the bit field as the value of the property BitField. For example, if you enter 2 as the value of BitField for an attribute named attribute_1 of type int, the resulting code is:

int attribute_1 : 2;

ConstantVariableAsDefine

The ConstantVariableAsDefine property is a Boolean value that determines whether the variable, defined as constant in file or package, is generated using a #define macro. Otherwise, it is generated using the const qualifier.

(Default = Checked)

DeclarationModifier

The property DeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in attribute declarations. Keywords that appear between the attribute type and the attribute name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and PostDeclarationModifier.

Default = Blank

DeclarationPosition

The DeclarationPosition property enables you to control the declaration order of attributes. The possible values are as follows:

- Default - Similar to the AfterClassRecord setting, with the following difference:
- For static attributes defined in a class with the property C_CG::Attribute::Visibility set to Public, these attributes are generated after types whose C_CG::Type::Visibility property is set to Public.
- You should not use this setting for new models.
- BeforeClassRecord - Generate the attribute immediately before the class record.
- AfterClassRecord - Generate the attribute immediately after the class record.
- StartOfDeclaration - Generate the attribute immediately after the start of the section (private or public

part of the specification, or package body).

- EndOfDeclaration - Generate the attribute immediately before the end of the section (private or public part of the specification, or package body).

(Default = Default)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

GenerateVariableHelpers

By default, Rational Rhapsody generates getter and setter methods for class attributes, but not for global variables. If you want Rational Rhapsody to generate getter and setter methods for global variables, set the value of the property GenerateVariableHelpers to True.

Default = Cleared

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by the software) to the beginning of the definition of a model element.

For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef_DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = empty MultiLine)

InitializationStyle

The InitializationStyle property specifies the initialization style used for attributes. When you specify an initial value for an attribute, Rational Rhapsody initializes the attribute based on the value of this property. In Rational Rhapsody Developer for C++, the possible values are as follows:

- ByInitializer - Initialize the attribute in the initializer (a(y)). This is the default value.
- If the initialization style is ByInitializer, the attribute initialization should be done after the user initializer, in the same order as the order of attributes in the code.
- ByAssignment - Initialize the attribute in the constructor body (a = y).

In Rational Rhapsody Developer for C, the attribute is initialized in the initializer body. (Default = ByInitializer)

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for C You can inline attribute and relation accessors and mutators for increased code performance. The visibility of the inlined operations can be either public or private. The accessor can contain only a return statement, or more than just a return statement.

For Rational Rhapsody Developer for C, there are two possible settings for this property:

none - The operation is not generated inline. This is the default. Example of "none": /* Mutator of Tank::ItsDishwasher relation */ void Tank_setItsDishwasher(struct Tank_t* const me, struct Dishwasher_t* p_Dishwasher) { if(p_Dishwasher != NULL) Dishwasher__setItsTank(p_Dishwasher, me); Tank__setItsDishwasher(me, p_Dishwasher); } /* Accessor to Tank::ItsDishwasher relation */ struct Dishwasher_t* Tank_getItsDishwasher(const struct Tank_t* const me) { return (struct Dishwasher_t*)me-itsDishwasher; }

The second possible setting is " in_header " to indicate that the operation is generated inline. Mutators are defined as macro definitions.

Example of "in_header": /* Inline Mutator of Tank::ItsDishwasher relation */ #define Tank_setItsDishwasher(me, p_Dishwasher) \ { \ if((p_Dishwasher) != NULL) \ Dishwasher__setItsTank((p_Dishwasher), (me)); \ Tank__setItsDishwasher((me), (p_Dishwasher)); \ }

Accessors that contain only return statements are defined as macros (the return statement and the semicolon at the end of expression are omitted); other accessors are generated as operations. For example: /* Inline Accessor to Tank::ItsDishwasher relation */ #define Tank_getItsDishwasher(me) ((me)-itsDishwasher)

If the attribute visibility is defined as Private, the macro definitions are placed in the implementation (.c) file.

If the attribute visibility is defined as Public, the macro definitions are placed in the specification (.h) file.

Note the following:

- Each instance of the macro’s parameters is parenthesized.
- You cannot inline an accessor if it contains statements other than the return statement. For example,

accessors to relations implemented using RiCCollection cannot be generated as function-like macros.

- You cannot set the Inline property separately for specific helpers (for example, only mutators) - this property affects all helpers of the attribute or relation.
- If a multi-lined mutator macro is called as the body of the “then” part of an “if ...else” statement, you must enclose it in parentheses or it generates a compilation error. For example:

```
// Erroneous code: If (itsDishwasher != NULL) Tank_setItsDishwasher(me, itsDishwasher); Else Return;
// Correct code: If (itsDishwasher != NULL) { Tank_setItsDishwasher(me, itsDishwasher); } Else Return;
```

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased.

(Default = Cleared)

IsMutable

The boolean property IsMutable allows you to specify that an attribute is a mutable attribute.

(Default = Cleared)

IsVolatile

The property IsVolatile allows you to specify that an attribute should be declared as volatile.

Default = Cleared

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations.

This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

(Default = common)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the

Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = Auto)

MutatorGenerate

The MutatorGenerate property specifies whether to generate mutators for attributes. The possible values are as follows:

- Smart - Mutators are not generated for attributes that have the Constant modifier.
- Always - Mutators are generated, regardless of the modifier.
- Never - Mutators are not generated.

(Default = Never)

MutatorVisibility

The MutatorVisibility property specifies the access level of the generated mutator for attributes. This enables you to define the access level of a mutator operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute's access level for the mutator.
- public - Set the mutator's access level to public.
- private - Set the mutator's access level to private.

- protected - Set the mutator’s access level to protected. This value is not available in Rational Rhapsody Developer for C.
- default - Set the mutator’s access level to default. This value is available only in Rational Rhapsody Developer for Java.

(Default = fromAttribute)

PreDeclarationModifier

The property PreDeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in attribute declarations. Keywords that appear before the attribute type are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties DeclarationModifier and PostDeclarationModifier.

Default = Blank

PostDeclarationModifier

The property PostDeclarationModifier is used to enable Rational Rhapsody to reverse engineer non-standard keywords that appear in attribute declarations. Keywords that appear after the attribute name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and DeclarationModifier.

Default = Blank

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code. (Default = "")*

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The

signatures of the two operations must match.

(Default = empty string)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rational Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for the element has been applied.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.

- For ImplementationEpilog, enter the value #endif

Default = Blank

VariableInitializationFile

The VariableInitializationFile property specifies how global constant variables are initialized. You can initialize these variables in the specification file. You can use these variables as compile-time constants that can be used to define array sizes, for example. Rational Rhapsody automatically identifies constant variables with const. By modifying this property, you can choose the initialization file directly. The possible values are as follows:

- Default - The variable is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize global constant variables in the implementation file.
- Specification - Initialize global constant variables in the specification file.

(Default = Default)

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The Visibility setting has the following applicability:

- Classes - Applies only to nested classes, which are defined inside other classes.
- Types - Applies only to types that are defined inside classes. It does not apply to global types, which are defined in packages.

Default = Protected

Class

The Class metaclass contains properties that affect the generated classes.

AccessTypeName

The AccessTypeName property specifies the name of the access type generated for the class record. (Default = empty string)

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active classes. The possible values are as follows:

- A string - Specifies the message queue size for an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

(Default = empty string)

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects. The possible values are as follows:

- Any integer - Specifies that a stack of that size is allocated for active objects.
- An empty string (blank) - If not specified, the stack size is set in an operating system-specific manner, based on the value of the ActiveStackSize property for the framework.

(Default = empty string)

ActiveThreadName

The ActiveThreadName property indicates the real OS task or thread name. This property only matters when the class is set to active. This facilitates debugging in complex environments in which many threads are constantly being created and deleted on-the-fly. This property is effective for all targets. All strings entered must be enclosed in quotes (" ").

(Default = NULL)

ActiveThreadPriority

The ActiveThreadPriority property specifies the priority of active class threads. The possible values are as follows:

- A string - Specifies thread priority of an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

(Default = empty string)

AdditionalBaseClasses

The AdditionalBaseClasses property enables you to add inheritance from external classes to the model.

(Default = empty string)

AdditionalNumberOfInstances

The AdditionalNumberOfInstances property is a string that specifies the size of the local heap allocated for events when the current pool is full. Triggered operations use the event properties. This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during runtime. All events are dynamically allocated during

initialization.

Once allocated, a thread’s event queue remains static in size. The possible values are as follows:

- An empty string (blank) - No additional memory is to be allocated when the initial memory pool is exhausted.
- n (a positive integer) - Specifies the size of the array allocated for additional instances.

(Default = empty string)

AllocateMemory

The AllocateMemory property specifies the string generated to allocate memory dynamically for objects or events. This string is used in the Create() operation. The default memory allocation string is as follows: ($cname *) malloc(sizeof($cname)); The variable $cname is replaced with the name of the object type during code generation. For example, the Create() operation generated for an object A uses this string to allocate memory for a new object as follows: A * A_Create(RiCTask * p_task) { A* me = (A *) malloc(sizeof(A)); A_Init(me, p_task); return me; } You can edit the memory allocation string to use a different mechanism than malloc(), if desired. The string used to free memory is specified with the FreeMemory property.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

BaseNumberOfInstances

The BaseNumberOfInstances property is a string that specifies the size of the local heap memory pool allocated for either:

- Instances of the class (C_CG::Class)
- Instances of the event (C_CG::Event)
- This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during run time. All instances of events are dynamically allocated during initialization. Once allocated, a thread’s event queue remains static in size.

Triggered operations use the properties defined for events. When the memory pool is exhausted, an additional amount, specified by the AdditionalNumberOfInstances property, is allocated. Memory pools for classes can be used only with the Flat statechart implementation scheme. The possible values are as follows:

- An empty string (blank) - Memory is always dynamically allocated.
- n (positive integer) - An array is allocated in this size for instances.

The related properties are as follows:

- AdditionalNumberOfInstances - Specifies the number of instances to allocate if the pool runs out.
- ProtectStaticMemoryPool - Specifies whether the pool should be protected (to support a multithreaded environment)
- EmptyMemoryPoolCallback - Specifies a user callback function to be called when the pool is empty. This property should be used instead of the AdditionalNumberOfInstance property for error handling.
- EmptyMemoryPoolMessage - When set to true, this property causes a message to be displayed if the pool runs out of memory in instrumented mode.

(Default = empty string)

ComplexityForInlining

The ComplexityForInlining property specifies the upper bound for the number of lines in user code that are allowed to be inlined. User code is the action part of transitions in statecharts. For example, using the value 3, all transitions with actions consisting of three lines or fewer of code are automatically inlined in the calling function. Inlining is replacing a function call in the generated code with the actual code statements that make up the body of the function.

This optimizes the code execution at the expense of an increase in code size. For example, increasing the number of lines that can be inlined from 3 to 5 has shortened the code execution time in some cases up to 10%. This property applies only to the Flat implementation scheme for statecharts. (Default = 0)

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to "MYDLL_API." The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class. (Default = empty string)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rational Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

Destructor

The Destructor property controls the generation of virtual destructors in C++. The property exists for C for historical reasons, with a single value of auto, but it has no effect on the generated C code. The possible values are as follows:

- auto - A virtual destructor is generated for an object only if it has at least one virtual function.
- virtual - A virtual destructor is generated in all cases.
- abstract - A virtual destructor is generated as a pure virtual function.
- common - A nonvirtual destructor is generated.

(Default = auto)

Embeddable

The Embeddable property is a Boolean property that specifies whether a class can be allocated by value (nested) inside another class or package.

For example, if the Embeddable property is Checked, 20 instances of a class A can be allocated inside

another class using the following syntax: A itsA[20]; The possible values are as follows:

- Checked - The object can be allocated by value inside a composite object or package. The object declaration and definition are generated in the specification file of the composite.
- Cleared - The object cannot be embedded inside another object (not supported in RiC). The object declaration and definition are generated in the implementation file of the composite.

The Embeddable property is used with the EmbeddedScalar and EmbeddedFixed properties to determine how to generate code for an embedded object. The Embeddable property must be set to True for either of those properties to take effect. It is also closely related to the ImplementWithStaticArray property, which also needs to be set in order to support by-value allocation. Relations can be generated by value only under the following circumstances:

- The multiplicity of the relation is well-defined (not “*”).
- The ImplementWithStaticArray property of the component relation is set to FixedAndBounded.

When the Embeddable property is False (RiC only):

- The attributes of the object are encapsulated. Clients of the object are forced to use it only via its operations, because there is no direct access to its attributes.
- Dynamic allocation must be used. The compiler does not know how to statically allocate an object when its declaration is not visible.

(Default = Checked)

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- Checked - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for the object or object type.
- Cleared - Events are dynamically allocated during initialization, but not during runtime. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during runtime.

(Default = Checked)

EnableUseFromCPP

The EnableUseFromCPP property specifies whether to wrap C operations with an appropriate extern C {} wrapper to prevent problems when code is compiled with a C++ compiler.

Wrapping C code with extern C enables you to include C code in a C++ application. Note that the structure definition for the object is not wrapped - only the functions are.

For example, if the EnableUseFromCPP is set to Checked for an object, the following wrapper code is generated for its operations:

#ifdef __cplusplus extern "C" { #endif /* __cplusplus */ /* Operations */ #ifdef __cplusplus } #endif /* __cplusplus */

(Default = Cleared)

Final

The Final property, when set to Cleared, specifies that the generated record for the class is a tagged record. This property applies to Ada95. (Default = Cleared)

FreeMemory

The FreeMemory property specifies the string generated to free memory previously allocated for objects or events. This string is used in the Destroy() operation. For an object, the free memory string is as follows: free($meName); The variable meName is replaced with the string used for the me context variable during code generation. For example, the Destroy() operation generated for an object A uses this string to free memory when an instance of A is destroyed as follows: void A_Destroy(A* const me) { A_Cleanup(me); free(me); } You can edit the string used to free memory to use a different mechanism than free(), if desired. The string used to allocate memory is specified with the AllocateMemory property. (Default = free($meName);)

GenerateAccessType

The GenerateAccessType property determines which access types are generated for the class. The possible values are as follows:

- None - Access types are not generated.
- Standard - An access type is generated.
- General - General access types are generated.

(Default = General)

GenerateDestructor

The GenerateDestructor property specifies whether to generate a destructor for a class. (Default = Checked)

GenerateRecordType

The GenerateRecordType property determines whether the class record is generated. (Default = Checked)

HasUnknownDiscriminant

The HasUnknownDiscriminant property determines whether an unknown discriminant >) is generated for this class. (Default = Cleared)

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces. (Default = empty string)>

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body. (Default = empty MultiLine)

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body. (Default = empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = empty MultiLine)

In

The In property specifies how code is generated when the type is used with an argument that has the modifier In. The C value "const $type*" is the default.

InitCleanUpRelations

The InitCleanUpRelations property specifies whether to generate initRelations() and cleanUpRelations() operations for sets of related global instances. This property applies only to composites and global relations. (Default = Checked)

InitializationCode

The InitializationCode property adds the specified initialization code in the body of the class. A non-abstract class can have initialization code that is executed during elaboration of the associated package. (empty MultiLine)

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier "InOut."

InstanceDeclaration

The InstanceDeclaration property specifies how instances are declared in code.

The default value for C is as follows: struct $cname$suffix In the generated code, the variable $cname is replaced with the object (or object type) name. The variable $suffix is replaced with the type suffix “_t,” if the object is of implicit type.

IsCompletedOperation

The IsCompletedOperation specifies whether state_IS_COMPLETED operations are generated as functions or macros (using #define). The possible values are as follows:

- Plain - state_IS_COMPLETED operations are generated as functions (pre-V4.2 behavior). This is the default value.
- Inline - state_IS_COMPLETED operations are generated using #define macros, if the body contains

only a return statement.

(Default = Plain)

IsInOperation

When Rational Rhapsody generates code for a class with a statechart, the code includes, by default, an _IN function for each state defined (for example, class_0_state_1_IN) whose return value indicates whether or not the system is currently in that state.

For statecharts that contain one or more And states, the generated code will include calls to these _IN functions.

The property IsInOperation can be used to control whether or not these _IN functions should be generated, and what the code looks like when they are generated. The possible values for the property are:

- Inline - Rational Rhapsody generates the _IN code as macros
- Plain - Rational Rhapsody generates the _IN code as functions
- Never - _IN functions are not generated for the states in the statechart

Note that if you set this property to Never, and your statechart includes one or more And states, the generated code will not compile because it will contain calls to non-existent functions.

Default = Plain

IsLimited

The IsLimited property determines whether the class or record type is generated as limited. (Default = Cleared)

IsNested

The IsNested property specifies whether to generate the class or package as nested. (Default = Cleared)

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private. (Default = Cleared)

IsReactiveInterface

The IsReactiveInterface property modifies the way reactive classes are generated. It has the following effects:

- Virtual inheritance from OMReactive
- Prevents instrumentation
- Prevents the thread argument and the initialization code (setting the active context) in the class

constructor

- Creates a pure-virtual destructor (by default)

This property affects only classes that declare themselves as interfaces by having a stereotype with a name that contains the word “interface” (case-insensitive). In previous versions of Rational Rhapsody, a class could inherit from a single reactive class only, regardless of whether it was an interface or implementation class. Beginning with Version 4.0.1 MR2, a class can inherit (implement) several reactive interfaces.

In Rational Rhapsody Developer for C++, you must explicitly designate reactive interfaces because the code generator applies special translation rules involving multiple inheritance from the Rational Rhapsody framework. You can designate a reactive interface in two ways:

- Set the property C_CG::Class::IsReactiveInterface to true.
- Use the predefined stereotype Reactive_interface. This stereotype uses stereotype-based code generation in order to automatically apply the correct property value.

Alternatively, you can define another stereotype (such as PortSpec) that sets IsReactiveInterface to true and use that stereotype. A class is considered reactive if it meets all the following conditions:

- The C_CG::Framework::ReactiveBase property is not empty.
- The C_CG::Framework::ReactiveBaseUsage property is set to true.
- One or more of the following conditions are true:
- The class has a statechart or activity diagram.
- The class is a composite class.
- The class has event receptions or triggered operations.

(Default = Checked)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#]

annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = Auto)

MaximumPendingEvents

The MaximumPendingEvents property specifies the maximum number of events that can be simultaneously pending in the event queue of the active class. The possible values are as follows:

- –1 - Memory is dynamically allocated.
- Positive integer - Specifies the maximum number of events.

(Default = –1)

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package. (Default = Public)

ObjectTypeAsSingleton

The ObjectTypeAsSingleton property enables you to generate singleton code for object-types and actors. This functionality enables you to save a singleton-type (actor) in its own repository unit, and manage that unit using a configuration management tool. Set this property for a single object-type or higher. An object-type is generated as a singleton when all of the following conditions are met:

- The object-type has the «Singleton» stereotype.
- There is one and only one object of the object-type and the object multiplicity is 1.
- The ObjectTypeAsSingleton property is set to True.

Note that when you expose a singleton object (for example, by creating a singleton object-type), Rational Rhapsody also modifies the code generated for the singleton. (Default = Cleared)

OptimizeStatechartsWithoutEventsMemoryAllocation

The OptimizeStatechartsWithoutEventsMemoryAllocation property determines whether the generated code uses dynamic memory allocation for statecharts that use only triggered operations. (Default = Cleared)

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out."

*(Default = $type**)*

PublishedName

This is the name that is used to identify the reactive object in order to send a distributed event to it. If there is only one reactive instance of the class, the value of this property is used to identify the object.

If there is more than one reactive instance of the class, each named explicitly, the name used to identify the reactive object is the name that you have given to the object, and not the property value.

In the case of multiplicity, where the objects are not named explicitly, the name used to identify the reactive object is the published name + the index of the object, for example, if the value of the property PublishedName is truck, then the objects would be identified by truck[0], truck[1]....

(Default = $name)

PublishInstance

This Boolean property indicates whether or not the object should be published as a reactive instance that is capable of receiving distributed events.

(Default = Cleared)

ReactiveThreadSettingPolicy

The ReactiveThreadSettingPolicy property enables you to specify how threads are set for reactive classes. The possible values are as follows:

- Default - During code generation, Rational Rhapsody adds a thread argument to the constructor.
- MainThread - Rational Rhapsody does not add an argument; the thread is set to the main thread.
- UserDefined - Rational Rhapsody does not add an argument; you must set the value for the thread yourself.

(Default = Default)

RecordTypeName

The RecordTypeName property specifies the name of the class record type. If this is not set, Rational Rhapsody uses class_name>_t. (Default = empty string)

RelativeEventDataRecordTypeComponentsNaming

The RelativeEventDataRecordTypeComponentsNaming property enables relative naming of event data

record type components that represent events and triggered operation parameters. If this is True, no events or triggered operations will share argument names because they would generate record components with the same name (which would not compile). (Default = Cleared)

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type.

(Default = $type)*

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rational Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for the element has been applied.

Default = "Default"

SimplifyConstructors

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyConstructors can be used to change the way constructors are handled by Rational Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The constructors is ignored.
- Copy - The constructors will just be copied from the original to the simplified model. They do not be modified in any way.
- Default - Uses the standard simplification for constructors, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for constructors has been applied.

Default = "Default"

SimplifyDestructors

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyDestructors can be used to change the way destructors are handled by Rational Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The destructors is ignored.
- Copy - The destructors will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for destructors, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for destructors has been applied.

Default = "Default"

SimplifyPackageFiles

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyPackageFiles can be used to change the way File elements are handled by Rational Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - File elements are ignored.
- Copy - File elements will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for File elements, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for File elements has been applied.

Default = "Default"

SingletonExposeThis

The SingletonExposeThis property, when set to False, specifies that all non-static methods are considered as static methods and will not have a this parameter passed in. (Default = Cleared)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification. (Default = empty MultiLine)

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification. (Default = empty MultiLine)

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces. (Default = empty string)

TaskBody

The TaskBody property enables you to define an alternate task body for Ada Task and Ada Task Type classes. (Default = empty string)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return." (Default = $type) See also:*

- In
- InOut
- Out

(Default = $type)*

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

(Default = Public)

Configuration

The Configuration metaclass contains properties for implementing configurations.

ClassStateDeclaration

The ClassStateDeclaration property supports C compilers that cannot handle enum declarations inside

struct declaration. The possible values are as follows:

- InClassDeclaration - Generate the reactive statechart enum declaration in the class declaration.
- BeforeClassDeclaration - Generate the reactive class statechart enum declaration before the declaration of the class.

Default = InClassDeclaration

CodeGenerationDirectoryLevel

The property CodeGenerationDirectoryLevel is found in the pre-72 compatibility profiles for C and C++.

Before version 7.2 of Rational Rhapsody, the directories specified with the properties DefaultSpecificationDirectory and DefaultImplementationDirectory were created at the beginning of the path to the generated files, for example, ..\spec_directory\package_a\subpackage_1 and ..\impl_directory\package_a\subpackage_1.

Beginning with version 7.2 of Rational Rhapsody, the directories specified with DefaultSpecificationDirectory and DefaultImplementationDirectory are created at the end of the path to the generated files, for example, ..\package_a\subpackage_1\spec_directory and ..\package_a\subpackage_1\impl_directory.

To provide the old code generation behavior for pre-72 models, the compatibility profiles include the property CodeGenerationDirectoryLevel, with the default value of the property set to Top. If you want your pre-72 models to use the new behavior that was introduced in version 7.2, change the value of this property to Bottom.

Default = Top

CodeGeneratorTool

The property CodeGeneratorTool specifies which code generation tool to use for the given configuration. The possible values are as follows:

- Advanced - Rational Rhapsody uses its internal code generator is used to generate code
- External - instructs Rational Rhapsody to use the registered external code generator
- Customizable - instructs Rational Rhapsody to use the customizable code generation mechanism.

If the property is set to Customizable, Rational Rhapsody carries out the following steps:

1. Creates a refined model from the original model. This model is referred to as the simplified model. (This step represents the transformation stage.)

2. Invokes the external RulesComposer code writer to create the code itself. (This step represents the writing stage.)

Note: The RulesComposer code writer is a .dll that is installed in the framework of the standard Rational Rhapsody installation. However, you can only use this writer if you have a valid license for using this feature.

Both of these steps (creation of the simplified model and generation of code from the simplified model) can be customized.

Default = Advanced

ContainerSet

The ContainerSet property specifies the container set used to implement relations.

(Default = RiCContainers)

CustomizableCG

The property CustomizableCG is deprecated as of Rational Rhapsody 7.2.

To specify that customized code generation should be used, set the value of the property C_CG::Configuration::CodeGeneratorTool to Customizable.

The description below was relevant prior to version 7.2 of Rational Rhapsody.

When you instruct Rational Rhapsody to generate code, Rational Rhapsody takes one of two different paths, depending on the value of the property C_CG::Configuration::CustomizableCG.

If CustomizableCG is set to Cleared, Rational Rhapsody starts its standard internal code generation mechanism. (This is the default setting for the property.)

If CustomizableCG is set to Checked, Rational Rhapsody carries out the following steps:

1. Creates a refined model from the original model. This model is referred to as the simplified model. (This step represents the transformation stage.)

2. Invokes the external RulesComposer code writer to create the code itself. (This step represents the writing stage.)

Note: The RulesComposer code writer is a .dll that is installed in the framework of the standard Rational Rhapsody installation. However, you can only use this writer if you have a valid license for using this feature.

Both of these steps (creation of the simplified model and generation of code from the simplified model) can be customized.

(Default = Cleared)

DefaultImplementationDirectory

The DefaultImplementationDirectory property specifies the relative path to the default directory for generated implementation files. The value of this property is added after the configuration path. Consider the following case:

- File C.cpp is an implementation of class C mapped to a folder Foo.
- The active configuration (cfg) is under component cmp.
- DefaultImplementationDirectory is set to “src”

Rational Rhapsody generates C.cpp to root>\cmp\cfg\src\Foo. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

(Default = empty string)

DefaultSpecificationDirectory

The DefaultSpecificationDirectory property specifies the relative path to the default directory for generated specification files. The value of this property is added after the configuration path. Consider the following case:

- File B.h is a specification of class B that is not mapped to any file.
- The active configuration (cfg) is under component cmp.
- DefaultSpecificationDirectory is set to “inc”

Rational Rhapsody generates B.h to root>\cmp\cfg\inc. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

(Default = empty string)

DependencyRuleScheme

The DependencyRuleScheme property specifies how dependency rules should be generated in the makefile. The possible values are as follows:

- Basic - Generates only the local implementation and specification files in the dependency rule in the makefile.
- ByScope - In addition to generating the same files as the Basic option, generates the specification files of related elements (dependencies, associations, generalizations, and so on) that are in the scope of the active component.
- Extended - In addition to generating the same files as the ByScope option, generates the specification files of related external elements (specified using the properties CG::Class/Package::UseAsExternal) and elements that are not in the scope of the active component.

(Default = ByScope)

DescriptionBeginLine

This property enables you to specify the prefix for the beginning of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation.

This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected.

(Default = /)*

DescriptionEndLine

This property enables you to specify the prefix for the end of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation. This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected.

*(Default = */)*

EmptyArgumentListName

The EmptyArgumentListName specifies the string generated for the argument list when an operation has no arguments. For example, if you set this value to "void," for an operation foo that has no arguments, Rational Rhapsody generates the following code:

int foo (void){...}

(Default = empty string)

Environment

The Environment property determines the target environment for a configuration. Generated code is targeted for that environment. See the Release Notes for the environments supported by Rational Rhapsody “out-of-the-box.”

“Out-of-the-box” support means that Rational Rhapsody includes a set of preconfigured code generation properties for the environment and precompiled versions of the relevant OXF libraries. The precompiled OXF libraries have been fully tested. You can also add new environments, for example if you want to generate code for another RTOS.

This involves retargeting the OS wrapper files in the Rational Rhapsody framework and creating a new set of code generation properties for the target environment.

(Default = Microsoft)

ExternalGenerationTimeout

The ExternalGenerationTimeout property specifies how long, in seconds, Rational Rhapsody waits for the each class in the configuration scope to complete so you can once again make changes to the model.

This property applies to both the full-featured external generator and makefile generator. For example, if you set this property to 2 and you have 10 classes, Rational Rhapsody sets a timeout of 20. If the external code generator does not complete generation in this timeframe, Rational Rhapsody displays a message in the output window saying that the generator is not responding, and you are allowed to make changes to the model.

If you set this property to 0, Rational Rhapsody does not time out the generation session, and waits for the code generator to complete its task - even if it takes forever. Rational Rhapsody waits for a notification from the full-featured external code generator, or for the process termination of a makefile generator.

(Default = 0)

ExternalGeneratorFileMappingRules

The ExternalGeneratorFileMappingRules property specifies whether the external code generator uses the same file mapping and naming scheme (mapping rules) as Rational Rhapsody. If the mapping rules are different, the external generator must implement handlers to the GetFileName, GetMainFileName, and GetMakefileName events that Rational Rhapsody runs to get a requested file name and path.

The possible values are as follows:

- AsRhapsody - The external generator uses the same mapping rules as Rational Rhapsody.
- DefinedByGenerator - The external generator has its own mapping rules.

(Default = AsRhapsody)

GeneratorExtraPropertyFiles

The GeneratorExtraPropertyFiles property launches the default Text Editor allowing the user to edit the $OMROOT\CodeGenerator\GenerationRules\LangC\RiC_CG.ini file.

GeneratorRulesSet

The GeneratorRulesSet property allows you to specify your own rule set to use for customized code generation.

Default = $OMROOT\CodeGenerator\GenerationRules\LangC\CompiledRules\RiCWriter.classpath

GeneratorScenarioName

The GeneratorScenarioName property specifies which scenario to use for the rule set, if you write your

own set of code generation rules.

Default = scenarios.Rhapsody_Generation.main

GenericEventHandling

The GenericEventHandling property is a Boolean value that determines whether to generate generic event-handling code. This property supports large-scale collaboration, where you might not be aware of which classes consume a base event of your part in the event hierarchy, and might not have access to parts of the model that use base events.

The framework base event class includes a virtual method that checks the event ID against the specified ID, thereby supplying a generic mechanism for events without super events. The language-specific methods are as follows: C:

#define RiCEvent_isTypeOf(event, id) ((event)-lId == (id)) C++: virtual OMBoolean isTypeOf(short id) const {return lId ==id;}

Each generated event that has a super event will override the method to check the ID against its own ID, then calls its base event directly to continue the check. An event without a base event will return Cleared if the ID does not equal its own.

When you set the GenericEventHandling property to Cleared, event consumption code is generated. Setting this property affects only the way events are consumed - the override on the isTypeOf() method is still generated, to allow handling of events in components that use the generic event handling. To support complete generic event handling, you should regenerate the code for all events and reactive classes.

(Default = Cleared)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rational Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = Empty MultiLine)

InitializeEmbeddableObjectsByValue

The InitializeEmbeddableObjectsByValue property specifies whether embeddable classes and object types selected in the configuration initial instances list should be allocated by value in the main() routine.

(Default = Cleared)

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation.

(Default = Empty MultiLine)

MainFunctionArgList

This property provides a list of the main function arguments. The default list is "int argc, char* argv[]."

Default = int argc, char argv[]*

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the

Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rational Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually.

The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters \n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = None)

MultipleAddressSpaces

When this boolean property is set to Checked, Rational Rhapsody uses the code generation settings required for use of the multiple address space feature.

Since the default value of this property is Cleared, you must change the value to enable this feature.

(Default = Cleared)

mxfCfgTemplate

The property mxfCfgTemplate serves as a template for generating the header file (by default named mxf_cfg.h) which is used for configuring the build of the MicroC framework (mxf).

You can include the values of various Rhapsody properties in the generated .h file, using either of the following two substitution mechanisms:

For properties located under C_CG::Configuration or under C_CG::<environment>, you can use the format $<property name> and Rhapsody will get the value of that property. For example, if the value of the property MyProp is 100, you can include a line like #define MY_FLAG $<MyProp> in the the template property, and in the generated .h file this is replaced by #define MY_FLAG 100.

For any Rational Rhapsody property that applies to the active configuration, you can use the format $<Subject.Metaclass.Property> and Rational Rhapsody gets the value of that property. For example, if the value of the property C_CG::Configuration::Environment is "Microsoft." you can include a line like #define MY_ENVIRONMENT $<C_CG.Configuration.Environment> in the template property, and in the generated .h file, this is replaced by #define MY_ENVIRONMENT Microsoft.

ShowCgSimplifiedModelPackage

The first step of the code generation process consists of the building of a simplified model based on the Rational Rhapsody model.

By default, the simplified model is not displayed in Rational Rhapsody. To have the simplified model displayed in the browser, set the property ShowCgSimplifiedModelPackage property to True. Once you have done so, the next time you generate code, the simplified model is added automatically at the top of the project tree in the browser.

Default = Cleared

SimplifyMainFiles

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyMainFiles can be used to change the way main files are handled by Rational Rhapsody when it transforms the model into a simplified model. This allows you to customize code generation for main files, beyond the initialization code you can specify in Rational Rhapsody at the configuration level.

The property can take any of the following values:

- None - Main files is ignored.
- Copy - Main files will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for main files, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for main files has been applied.

Default = "Default"

SimplifyMakeFile

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyMakeFile can be used to change the way makefiles are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Makefile elements are ignored.
- Copy - Makefile elements will just be copied from the original to the simplified model. They will not be modified in any way.

- Default - Uses the standard simplification for makefiles, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for makefiles has been applied.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

Cygwin

The Cygwin metaclass controls the environment settings (Compiler, framework libraries, etc.) for Cygwin.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/Cygwin)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rational Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

AnimIncludeDirectories

The property AnimIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangC/aom $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangC/lib/cygwinaomanim$(LIB_EXT)

AnimOxfLibs

The property AnimOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangC/lib/cygwinoxfinst$(LIB_EXT)

$(OMROOT)/LangC/lib/cygwinomcomappl$(LIB_EXT)

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMANIMATOR $(DEFINE_QUALIFIER)__USE_W32_SOCKETS

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = $makefile

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = cygwinmake.bat

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag

(:cx_option=exceptions).

- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\"\etc\cygwinmake.bat cygwinbuild.mak "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Checked

CompileRelease

The CompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

CompilerFlags

The property CompilerFlags allows you to define additional compilation flags. The value of the property is inserted into the value of the property CompileSwitches (Linux) or CPPCompileSwitches (cygwin). In the generated makefile, you can see the value of this property in the line that begins with ConfigurationCPPCompileSwitches=.

Default = Blank

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component. The default values are as follows:

(Default = -g)

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = -O)

CPPCompileSwitches

The CPPCompileSwitches property specifies the compiler switches.

The default value is as follows:

$IncludeDirectories $DefinedSymbols $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) $CompilerFlags $OMCPPCompileCommandSet -c

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default value is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Checked

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

You may also want to use the "Filter" facility in this window to refer to the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property.

ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is as follows:

ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

This default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)\LangC\lib\cygwinWebComponents$(LIB_EXT), $(OMROOT)\lib\cygwinWebServices$(LIB_POSTFIX)$(LIB_EXT), -lws2_32.

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

The default value is "include."

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\cygwinrun.bat\" $executable")

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\cygwinmake.bat\" $makefile $maketarget")

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = -g)

LinkerFlags

The property LinkerFlags allows you to define linker flags. The value of the property is inserted into the value of the property LinkSwitches. In the generated makefile, you can see the value of this property in the line that begins with LINK_FLAGS=.

Default = Blank

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

(Default = -O)

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

(Default = $OMLinkCommandSet $LinkerFlags)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release)

################ ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT)

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################# Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NoneIncludeDirectories

The property NoneIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = Blank

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = Blank

NoneOxfLibs

The property NoneOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangC/lib/cygwinoxf$(LIB_EXT)

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile. The default values are as follows:

Environment Default Value INTEGRITY work Integrity ESTL MultiWin32 obj_dir All others Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Checked)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error|warning)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make

process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (make)[:](.)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (error|undefined|cannot find|multiple definition)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host.

You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

The extension ".h" is the default for C.

TraceIncludeDirectories

The property TraceIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangC/aom $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangCpp/lib/cygwintomtraceRiC$(LIB_EXT) $(OMROOT)/LangCpp/lib/cygwinomcomappl$(LIB_EXT) $(OMROOT)/LangCpp/lib/cygwinoxf$(LIB_EXT) $(OMROOT)/LangC/lib/cygwinaomtrace$(LIB_EXT)

TraceOxfLibs

The property TraceOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangC/lib/cygwinoxfinst$(LIB_EXT) $(OMROOT)/LangC/lib/cygwinomcomappl$(LIB_EXT)

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMTRACER

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style.

If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

(Default = Checked)

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

(Default = Cleared)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

Dependency

The Dependency metaclass controls the dependency for a package that defines a namespace.

CreateUseStatement

The CreateUseStatement property determines whether a use statement is added to the code after the with statement. The supplier of the dependency must be a class or type. (Default = Cleared)

GenerateOriginComment

When set to True, generates a comment before #include statements indicating which element "caused" the #include.

GeneratePragmaElaborate

The GeneratePragmaElaborate property determines whether to generate an elaborate pragma for the supplier class in the client class or package. (Default = Cleared)

GeneratePragmaElaborateAll

The GeneratePragmaElaborateAll property determines whether to generate a pre-elaborate pragma for the supplier class in the client class or package. (Default = Cleared)

GenerateWithClause

The GenerateWithClause property determines whether with clauses are generated for Usage dependencies. For example, you can generate a with clause for a package, P1, in the specification of another package, P2, using a dependency, D1, and generate a use clause for P1 in the body of P2. In addition, this functionality is useful for modeling inherited annotations across classes and packages. (Default = Checked)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.

- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = empty MultiLine)

IncludeStyle

The IncludeStyle property controls the style of #include statements. Using this property, you can control the style of a specific dependency, or the entire configuration/component/project. To set the style for include files that are synthesized based on associations between model elements (for example, setting the type of some attribute to a class), add a «Usage» dependency between the elements and set this property to the appropriate value. The possible values are as follows:

- Default - Use angle brackets for include statements for external elements, and quotes for include statements for other elements.
- Quotes - Enclose include files in quotation marks. For example: #include "A.h"
- When a compiler encounters an include file in quotes, it searches for the file in both the current directory and the directories specified in the include path. Note that the specific algorithm used is compiler-dependent.
- AngledBrackets - Enclose include files in angle brackets. For example: #include A.h
- When a compiler encounters an include file in angle brackets, it searches for the file only in the directories specified in the include path.
- If you set the property to AngledBrackets at the configuration level, you must also change the

CG::File::IncludeScheme property to RelativeToConfiguration to ensure successful compilation.

(Default = Default)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = Auto)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)

- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for the element has been applied.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

UseNameSpace

The UseNameSpace property enables you to model namespace usage. When you set a dependency to a package that defines a namespace and set this property to True, Rational Rhapsody generates a “using namespace” statement to the package namespace. (Default = Cleared)

Event

The Event metaclass contains properties that control events.

AllocateMemory

The AllocateMemory property specifies the string generated to allocate memory dynamically for objects or events. This string is used in the Create() operation. The default memory allocation string is as follows: ($cname *) malloc(sizeof($cname)); The variable $cname is replaced with the name of the object type during code generation. For example, the Create() operation generated for an object A uses this string to allocate memory for a new object as follows: A * A_Create(RiCTask * p_task) { A* me = (A *) malloc(sizeof(A)); A_Init(me, p_task); return me; } You can edit the memory allocation string to use a different mechanism than malloc(), if desired. The string used to free memory is specified with the FreeMemory property.

AnimInstanceCreate

The AnimInstanceCreate property affects event creation. If you set the C_CG::Event::NoDynamicAllocAnimCreate property to False, Rational Rhapsody does not generate the event creation method, effectively disabling the ability to inject the event in animation. To enable the injection of the event, you can specify a different method to obtain an instance of the event by setting this property to the name of the method to use.

(Default = empty string)

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows:

class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to “MYDLL_API.” The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class.

(Default = empty string)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type

$Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified element’s tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- Checked - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for the object or object type.
- Cleared - Events are dynamically allocated during initialization, but not during run time. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during run time.

If you are managing your own memory pools, set this property to False and call CPPReactive_gen() directly. The following example shows how to call RiCReactive_gen() directly to send a static event to a reactive object A, when using a member function of A genStaticEv2A():

void A_genStaticEv2A(struct A_t* const me) { { /*#[operation genStaticEv2A() */ static struct ev _ev; ev_Init(_ev); RiCEvent_setDeleteAfterConsume(((RiCEvent*)_ev), RiCFALSE); (void) RiCReactive_gen(me-ric_reactive, ((RiCEvent*)_ev), RiCFALSE); /*#]*/ } }

Alternatively, you can use internal memory pools by setting the property BaseNumberOfInstances, which results in the use of framework memory pools. If you use the framework memory pools, do not disable the Create() and Destroy() methods because these methods are used to manage the memory pool. When you disable the generation of the Create() and Destroy() methods, you can still inject events in animation by supplying an alternate function to get an event instance.

To do this, set the AnimInstanceCreate property.

(Default = Checked)

FreeMemory

The FreeMemory property specifies the string generated to free memory previously allocated for objects or events. This string is used in the Destroy() operation. For an object, the free memory string is as follows: free($meName); The variable meName is replaced with the string used for the me context variable during code generation. For example, the Destroy() operation generated for an object A uses this string to free memory when an instance of A is destroyed as follows: void A_Destroy(A* const me) { A_Cleanup(me); free(me); } You can edit the string used to free memory to use a different mechanism than free(), if desired. The string used to allocate memory is specified with the AllocateMemory property. (Default = free($meName);)

In

The property In determines the exact syntax used when an event is used as an "in" parameter for an operation.

*Default = const $type**

InOut

The property InOut determines the exact syntax used when an event is used as an "in/out" parameter for an operation.

*Default = $type**

Out

The property Out determines the exact syntax used when an event is used as an "out" parameter for an operation.

*Default = $type***

ReturnType

The property ReturnType determines the exact syntax used when an event is used as the return type of an operation.

*Default = $type**

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for the element has been applied.

Default = "Default"

File

The File metaclass contains properties that control the generated code files.

DiffDelimiter

The DiffDelimiter property defines a symbol that is used to avoid overwriting an unchanged line of code during code generation. Use this property to avoid touching the source code file when the “diff-delimited” line has not changed. In general, fewer source files need to be recompiled if fewer source files are touched. For example, the DiffDelimiter symbol “//!” is used in the C_CG::File::Header property. This symbol is at the beginning of a line of code that includes the current code generation date.

The code generator compares the code it would normally generate for that line (the current code generation date) to that previously generated (the last code generation date).

If the date has not changed, the line is not overwritten, possibly preventing the file’s modification time from changing (being "touched").

(Default = //!)

Footer

The Footer property specifies a multiline footer that is added to the end of generated Java files. The default footer template is as follows:

"/********** File Path: $FullCodeGeneratedFileName **********/"

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.

- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property C_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

Header

The Header property specifies a multiline header that is added to the top of all generated Java files. The default header template is as follows:

/********** Rhapsody : $RhapsodyVersion Login : $Login Component : $ComponentName Configuration : $ConfigurationName Model Element : $FullModelElementName //! Generated Date : $CodeGeneratedDate File Path : $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property C_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”.

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementationFooter

The ImplementationFooter property specifies the multiline footer to be generated at the end of implementation files. The default footer template for C is as follows:

/********** File Path: $FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.

- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property C_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

Keyword names can be written in parentheses. For example: $(Name)

If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

ImplementationHeader

The ImplementationHeader property specifies the multiline header that is generated at the beginning of implementation files. The default header template for C is as follows:

/********** Rhapsody: $RhapsodyVersion Login: $Login Component: $ComponentName Configuration: $ConfigurationName Model Element: $FullModelElementName //! Generated Date: $CodeGeneratedDate File Path: $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.

- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property C_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = empty MultiLine)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = Auto)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user.
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after the Rational Rhapsody standard simplification for the element has been applied.

Note that this property refers to the simplification of component files.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when

the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationFooter

The SpecificationFooter property specifies the multiline footer to be generated at the end of specification files. The default footer template for C is as follows:

/********** File Path: $FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property C_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)

- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

SpecificationHeader

The SpecificationHeader property specifies the multiline header to be generated at the beginning of specification files. The default header template for C is as follows:

/********** Rhapsody: $RhapsodyVersion Login: $Login Component: $ComponentName Configuration: $ConfigurationName Model Element: $FullModelElementName //! Generated Date: $CodeGeneratedDate File Path: $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified the element tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property C_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class Yes No Inside Package Yes Yes Inside

(Default = empty MultiLine)

Framework

The Framework metaclass contains properties that affect the Rational Rhapsody framework.

ActivateFrameworkDefaultEventLoop

The ActivateFrameworkDefaultEventLoop property specifies the framework call that initializes the framework main event loop. (Default = OXF::start($Fork);) The value of $Fork is calculated from the property CG::Configuration::StartFrameworkInMainThread for regular applications and from the property CORBA::Configuration::StartFrameworkInMainThread for CORBA servers. This property can be set at the configuration level or higher.

ActiveBase

The ActiveBase property specifies the superclass from which to specialize all threads, if the ActiveBaseUsage property is set to True.

(Default = RiCTask)

ActiveBaseUsage

The ActiveBaseUsage property specifies whether to use the superclass specified by the ActiveBase property as the superclass for all threads.

(Default = Checked)

ActiveDestructorGuard

The ActiveDestructorGuard property specifies the macro that starts protection for an active user object destructor.

(Default = START_DTOR_THREAD_GUARDED_SECTION)

ActiveExecuteOperationName

The ActiveExecuteOperationName property sets the user object virtual table for an active object and passes it to a task in the task initialization function (RiCTask_init()). Follow these steps:

- Create a method with the following signature: struct RiCReactive * operation name> (RiCTask * const)
- Set the operation name in the ActiveExecuteOperationName property.
- Start the execution of the active object task by calling the RICTASK_START() macro on the object.

The virtual function table member name is stored in the ActiveVtblName property. (Default = empty string)

ActiveGuardInitialization

The ActiveGuardInitialization property specifies the call that makes the active object event dispatching guarded.

(Default = SetToGuardThread)

ActiveIncludeFiles

The ActiveIncludeFiles property specifies the base class for threads when using selective framework includes. If a class is active and this property is defined, the file specified by the property is included in the class specification file.

The default value for C is oxf/RiCTask.h.

ActiveInit

The ActiveInit property specifies the format of the declaration generated for the initializer for an active class.

The default value for C is $base_init($member, RiCFALSE, $Vtbl) .

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active objects, if the ActiveMessageQueueSize property for classes is left blank. The default value for the size of the message queue is language-dependent, as follows:

- C - The default value is RiCOSDefaultMessageQueueSize, which is a variable set in the implementation file of the OS adapter for a given operating system.
- C++ - The default value is OMOSThread::DefaultMessageQueueSize.
- Java - The default value is an empty string (blank).

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects, if the ActiveStackSize property for classes is left blank. The default value for the stack size is language-dependent, as follows:

- C - The default value is RiCOSDefaultStackSize, which is a variable set in the implementation file of the OS adapter for a given operating system.
- C++ - The default value is OMOSThread::DefaultStackSize.
- Java - The default value is an empty string (blank).

ActiveThreadName

The ActiveThreadName property specifies the name of threads, if the ActiveThreadName property for classes is left blank. The default values are as follows:

- A string - Names the active thread.
- NULL - The value is set in an operating system-specific manner, based on the value of the ActiveThreadName property for the framework.

(Default = NULL)

ActiveThreadPriority

The ActiveThreadPriority priority specifies the priority of threads, if the ActiveThreadPriority property for classes is left blank. The default value for the priority of threads is language-dependent, as follows:

- C - The default value is RiCOSDefaultThreadPriority, which is a variable set in the implementation file of the OS adapter for a given operating system.
- C++ - The default value is OMOSThread::DefaultThreadPriority.
- Java - The default value is an empty string (blank).

ActiveVtblName

The ActiveVtblName property stores the name of the virtual function table associated with a task (the

RiCTask member of the structure). (Default = $objectName_activeVtbl)

BooleanType

The BooleanType property specifies the Boolean type used by the framework. (Default = bool)

CurrentEventId

The CurrentEventId property specifies the call or macro used to obtain the ID of the currently consumed event.

(Default = OM_CURRENT_EVENT_ID)

DefaultProvidedInterfaceName

The DefaultProvidedInterfaceName property specifies the interface that must be implemented by the “in” part of a rapid port. See the Rational Rhapsody Help for more information on rapid ports.

(Default = DefaultProvidedInterface)

DefaultReactivePortBase

The DefaultReactivePortBase property stores the base class for the generic rapid port (or default reactive port). This base class relays all events. See the Rational Rhapsody Help for more information on rapid ports.

(Default = RiCDefaultReactivePort)

DefaultReactivePortIncludeFiles

The DefaultReactivePortIncludeFiles property specifies the include files that are referenced in the generated file that implements the class with the rapid ports. See the Rational Rhapsody Help for more information on rapid ports. (Default = oxf/OMDefaultReactivePort.h)

DefaultRequiredInterfaceName

The DefaultRequiredInterfaceName property specifies the interface that must be implemented by the “out” part of a rapid port. See the Rational Rhapsody Help for more information on rapid ports. (Default = DefaultRequiredInterface)

EnableDirectReactiveDeletion

The EnableDirectReactiveDeletion property specifies the call to the framework that supports direct deletion of reactive instances (using the delete operator) instead of graceful framework termination (using

the reactive destroy() method). When using destroy(), the object waits in a zombie mode until all the events that are designated to it are removed from the active context queue, and then self -destructs.

In this scheme, there is no need to traverse the queue of the active context to cancel pending events, and there is no need to make the reactive destructor guarded to ensure safe deletion. A reactive object can be either in a graceful termination or forced deletion (using the delete operator) state: you cannot use graceful deletion on an object that allows forced deletion, and vice versa. You can set a single reactive object in a forced deletion state, or set the entire system (all reactive instances) in a forced deletion state (as is done for backward compatibility).

Graceful termination should not be used when a reactive part (of a composite class) runs in a context of an active object that is not part of, and different from, the composite active context. If you are using a Rhapsody library component as part of an application where the main is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init().

If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call. Note that the property C_CG::Framework::UseDirectReactiveDeletion must be set to True for this property to take effect. When it is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init().

(Default = OXF::supportExplicitReactiveDeletion();)

EventBase

The EventBase property specifies the base class for all events.

(Default = RiCEvent)

EventBaseUsage

The EventBaseUsage property specifies whether to use the event superclass specified by the EventBase property as the parent of all events.

The C default value is Checked.

EventGenerationPattern

The EventGenerationPattern property supplies some of the information needed to generate code for Send Action elements in C, C++, and Java.

For each language, code generation for this element is determined by the following properties:

- C_CG::Framework::EventGenerationPattern - general format
- C_CG::Framework::EventToPortGenerationPattern - used when sending even to a port

Note:

Rhapsody does not support roundtripping for Send Action elements.

(Default = RiCGEN($meArrow$target, $event))

EventIncludeFiles

The EventIncludeFiles property specifies the base class for events when using selective framework includes. If events are defined in a package, the file specified by this property is included in the package specification file to enable the use of events and timeouts in the package.

The default value for C is oxf/RiCEvent.h.

EventSetParamsStatement

The EventSetParamsStatement property specifies a template for the body of the setParams() method, provided by the Rational Rhapsody framework for Java, to set the parameters of an event. For example, for an event of type evOn(), the default template would generate the following code in the body of the setParams() method: evOn params = (evOn) event; The default value is as follows: $eventType params = ($eventType) event;

FrameworkInitialization

The FrameworkInitialization property specifies the framework initialization code that is called by the main. The default value is as follows: OXF::initialize($(Argc)$(Argv)$(AnimationPortNumber) $(RemoteHost)$(TimerResolution)$(TimerMaxTimeouts) $(TimeModel)))

HeaderFile

The HeaderFile property specifies the framework header files to be included in objects that are within the scope of a particular configuration. The default values are as follows:

Default Generated Statement "oxf/Ric.h" #include oxf/Ric.h

To optimize your code for size, leave the HeaderFile property blank. In this way, you can explicitly include the framework only when needed.

IncludeHeaderFile

The IncludeHeaderFile property specifies whether to include the framework header files specified by the CG::Framework::HeaderFile property in the project.

The C default value is Checked.

InnerReactiveClassName

The InnerReactiveInstanceName property enables you to specify the name of a reactive class that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive.

The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance. (Default = Reactive)

InnerReactiveInstanceName

The InnerReactiveInstanceName property enables you to specify the name of a reactive instance that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive.

The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance. (Default = reactive)

InstrumentVtblName

The InstrumentVtblName property specifies the name of the virtual function table associated with animation objects. Each animated object has its own virtual function table (Vtbl). This table enables you to create your own framework, with its own virtual instrumentation functions, and connect it to Rational Rhapsody. (Default = $objectName_instrumentVtbl)

IsCompletedCall

The IsCompletedCall property specifies the call or macro that determines whether the state reached a final state so it can be exited on a null transition. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name. (Default = IS_COMPLETED($State))

IsInCall

The IsInCall property specifies the query that determines whether the state is in the current active configuration. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name. (Default = IS_IN($State))

MakeFileName

The MakeFileName property enables you to specify a new name for the makefile. To use this property, add the following line to the .prp file:

Property MakeFileName String "MyFileName"

In this syntax, MyFileName specifies the name of the makefile.

NullTransitionId

The NullTransitionId property specifies the ID reserved for null transition consumption. (Default = OMEventNullId)

OperationGuard

The OperationGuard property specifies the macro that guards an operation. (Default = GUARD_OPERATION)

ProtectedBase

The ProtectedBase property specifies the base class for protected objects, if the ProtectedBaseUsage property is set to True.

The C default value is RiCMonitor.

ProtectedBaseUsage

The ProtectedBaseUsage property specifies whether to use the class specified by the ProtectedBase property as the base class for protected objects.

The C default value is Checked.

ProtectedClassDeclaration

The ProtectedClassDeclaration property affects how protected classes are implemented. Beginning with Rational Rhapsody 4.0, instead of inheriting from OMProtected, the class embeds an aggregate OMProtected. The aggregate member and helper methods are defined in the macro OMDECLARE_GUARDED (defined in omprotected.h).

(Default = OMDECLARE_GUARDED)

ProtectedIncludeFiles

The ProtectedIncludeFiles property specifies the base class for protected classes when using selective framework includes. The default value for C is as follows: oxf/RiCProtected.h

ProtectedInit

The ProtectedInit property specifies the declaration generated for the initializer for guarded objects.

The default value for C is $base_init($member).

ReactiveBase

The ReactiveBase property specifies the base class for all reactive classes, if the ReactiveBaseUsage property is set to True.

ReactiveBaseUsage

The ReactiveBaseUsage property specifies whether to use the class specified by the ReactiveBase property as the base class for all reactive objects. The default value is Checked.

ReactiveConsumeEventOperationName

The ReactiveConsumeEventOperationName property sets the user object virtual table for a reactive object. Follow these steps:

- Create a method with the following signature: void operation name>(RiCReactive * const, RiCEvent*)
- Set the operation name in the ReactiveConsumeEventOperationName property.

Rational Rhapsody Developer for Ada ignores all the values for the properties under the Framework metaclass except for this one. (Default = empty string)

ReactiveCtorActiveArgDefaultValue

The ReactiveCtorActiveArgDefaultValue property specifies the default value of the active context argument in a reactive constructor. (Default = 0)

ReactiveCtorActiveArgName

The ReactiveCtorActiveArgDefaultValue property specifies the name of the active context argument in a reactive constructor. (Default = activeContext)

ReactiveCtorActiveArgType

The ReactiveCtorActiveArgDefaultValue property specifies the type of the active context argument in a reactive constructor. (Default = IOxfActive)*

ReactiveDestructorGuard

The ReactiveDestructorGuard property specifies the macro that starts protection of a section of code used for destruction of a reactive instance. This prevents a “race” (between the deletion and event dispatching)

when deleting an active instance. (Default = START_DTOR_REACTIVE_GUARDED_SECTION)

ReactiveEnableAccessEventData

The ReactiveEnableAccessEventData property specifies the code to be used to enable access to the specific event data in a transition (typically by assigning a local variable of the appropriate type). The property supports the $Event keyword so you can specify the event type. (Default = RiCSETPARAMS($me, $Event);)

ReactiveGetStateCall

The ReactiveGetStateCall property is used for serialization to define the prototype of the getState framework method.

(Default = RiCReactive_getState(&(me->ric_reactive));)

ReactiveGuardInitialization

The ReactiveDestructorGuard property specifies the framework call that makes the event consumption of a specific reactive class guarded. (Default = setToGuardReactive)

ReactiveHandleEventNotConsumed

The ReactiveHandleEventNotConsumed property registers a method to handle unconsumed events in a reactive class. Specify the method name as this property's value. (Default = empty string)

ReactiveHandleTONotConsumed

The ReactiveHandleTONotConsumed property registers a method to handle unconsumed trigger operations in a reactive class. Specify the method name as this property's value. (Default = empty string)

ReactiveIncludeFiles

The ReactiveIncludeFiles property specifies the base classes for reactive classes when using selective framework includes. If a class is reactive and this property is defined, the file specified by the property is included in the class specification file. For reactive classes, the header files specified by the following properties are also included:

- EventIncludeFiles - For the event base class
- ActiveIncludeFiles - If the class is guarded or instrumented

The default value for C is oxf/RiCReactive.h.

ReactiveInit

The ReactiveInit property specifies the declaration for the initializer generated for reactive objects.

The default pattern for C is as follows: $base_init($member, (void*)$mePtr, $task, $VtblName);

The $base variable is replaced with the name of the reactive object during code generation.

The string "_init" is appended to the object name in the name of the operation. For example, if the reactive object is named A, the initializer generated for A is named A_init(). The $member variable is replaced with the name of the reactive member (equivalent to the base class) of the object during code generation.

The $mePtr variable is replaced with the name of the user object (the value of the Me property). The member and mePtr objects are not equivalent if the user object is active.

The $VtblName variable is replaced with the name of the virtual function table for an object, specified by the ReactiveVtblName property.

ReactiveInterface

The ReactiveInterface property specifies the name of the interface class that forwards messages to an inner class instance of a reactive class in order to implement its reactive behavior. (Default = RiJStateConcept)

ReactiveSetEventHandlingGuard

The ReactiveSetEventHandlingGuard property enables you to control the code generated within the constructor of a reactive class. When you use this property with guarded triggered operations, it enables guarding of the event handling (in order to provide mutual exclusion between the event and TO handling). (Default = setEventGuard(getGuard());)

ReactiveSetStateCall

The ReactiveGetStateCall property is used for serialization to define the prototype of the setState framework method.

The C Default is RiCReactive_setState(&(me->ric_reactive), oxfReactiveState);.

ReactiveSetTask

The ReactiveSetTask property specifies the string that tells a reactive object whether it is an active or a sequential instance. The default value for Ada is an empty string. The default value for C is as follows: RiCReactive_setActive($member, $isActive);

ReactiveStateType

The ReactiveStateType property is used for serialization to define the oxfstate type.

(Default = long)

ReactiveVtblName

The ReactiveVtblName property specifies the name of the virtual function table (Vtbl) associated with a reactive object. Each reactive object has its own Vtbl, which enables you to create your own framework and connect it to Rational Rhapsody. (Default = $objectName_reactiveVtbl)

SetManagedTimeoutCanceling

The SetManagedTimeoutCanceling property is a property for backward compatibility that specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (where OMTimerManager is responsible for cancellation of timeouts) or the Rational Rhapsody 6.0 scheme. In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object).

This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling).

If you are using a Rhapsody library component as part of an application where the main is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init(). If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call.

(Default = OXF::setManagedTimeoutCanceling(true);)

SetRhp5CompatibilityAPI

The SetRhp5CompatibilityAPI property specifies the call that configures models created before Rhapsody 6.0 so they use the 5. x version of the framework instead of the new one. You may also want to use the "Filter" facility in this window to refer to the definition of UseRhp5CompatibilityAPI for more information on Version 5. x compatibility mode. (Default = OXF::setRhp5CompatibleAPI(true);)

StaticMemoryIncludeFiles

The StaticMemoryIncludeFiles property specifies the files to be included in the package specification file if static memory management is enabled and you are using selective framework includes. (Default = oxf/MemAlloc.h)

StaticMemoryPoolDeclaration

The StaticMemoryPoolDeclaration property specifies the declaration of the memory pool for timeouts. The default value is as follows: DECLARE_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances)

StaticMemoryPoolImplementation

The StaticMemoryPoolImplementation property specifies the generated code in the implementation file for a memory pool implementation (see the BaseNumberOfInstances property). The default value is as follows:

IMPLEMENT_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances, $AdditionalNumberOfInstances, $ProtectStaticMemoryPool)

TestEventTypeCall

The TestEventTypeCall property specifies the test used in event consumption code to check if the currently consumed event is of a given type. (Default = IS_EVENT_TYPE_OF($Id))

TimeoutId

The TimeoutId property specifies the ID reserved for timeout events. (Default = OMTimeoutEventId)

TimerMaxTimeouts

The TimerMaxTimeouts property specifies the maximum number of timeouts allowed simultaneously in the system, if the TimerMaxTimeouts property for the configuration is not overridden. In the framework, the default number of timers is 100. (Default = empty string)

TimerResolution

The property TimerResolution allows you to override the default tick time used.

The number entered is the number of milliseconds used for the tick time.

The default tick time (currently 100 milliseconds) is defined by RiCTimerManagerDefaultTicktime in the file RiCTimer.c

Default = Blank

UseDirectReactiveDeletion

The UseDirectReactiveDeletion property determines whether direct deletion of reactive instances (using the delete operator) is used instead of graceful framework termination (using the reactive destroy() method). When this property is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init(). See the EnableDirectReactiveDeletion property definition and the upgrade history on the support site for more information on this functionality.

(Default = Cleared)

UseManagedTimeoutCanceling

The UseManagedTimeoutCanceling property specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (so OMTimerManager is responsible for cancellation of timeouts). In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object).

This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling). When loading a pre-6.0 model, Rational Rhapsody sets the project C_CG::Framework::UseManagedTimeoutCanceling to True to set the system-compatibility mode. See the upgrade history on the support site for more information.

(Default = Cleared)

UseRhp5CompatibilityAPI

The UseRhp5CompatibilityAPI property specifies whether to use the virtual functions of the core implementation classes that existed in the pre-Rhapsody 6.0 framework. The Rhapsody 6.0 framework introduces a set of interfaces for the core behavioral framework.

The interfaces define a concise API for the framework and enable you to replace the actual implementation of these interfaces while maintaining the framework behavior. As a result of the interfaces' introduction, the framework behavioral classes (OMReactive, OMThread, and OMEvent) use a new set of virtual operations to implement the interfaces and provide the behavioral infrastructure. To support existing customizations of these classes (made by inheriting and overriding the virtual operations), the framework can work in a mode where the pre-6.0 API virtual operations are called.

When loading a pre-6.0 model, Rational Rhapsody sets the project property C_CG::Framework::UseRhp5CompatibilityAPI to True to set the system-compatibility mode. If this is set to True, the pre-6.0 API is called by the framework instead of the interface-based API. Without this flag, user customizations will compile but will not be called. See the upgrade history on the support site for more information on the Version 5. x compatibility mode. (Default = Cleared)

Generalization

The Generalization metaclass contains a property used to support generalization. See the Rational Rhapsody Help for more information on generalization.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for the element has been applied.

Default = "Default"

INTEGRITY

This metaclass contains the properties that manipulate the INTEGRITY operating system environment.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

This property reflects the change in Version 4.1 where the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created.

This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment. The default values are as follows:

(Default = empty string)

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags. The default values are as follows:

(Default = empty string)

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches. The C INTEGRITY default value is as follows:

:optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

(Default = empty MultiLine)

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model. The default value for Ada is as follows:

:target_os=integrity :C_library=full :integrity_option=dynamic :staticlink=true

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

The C default value is as follows:

:defines=_DEBUG :target_os=integrity

BLDTarget

The BLDTarget property specifies the target BSP (e.g., ":target=Win32"). This property also affects the names of the framework libraries used in the link. The C default value is "mbx800."

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag

(:cx_option=exceptions).

- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

The default value for MultiWin32 is DebugNoExp; for the other environments, the default value is Debug.

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" IntegrityBuild.bat buildLibs bld $BLDTarget"

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

The default value is "main."

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default values are as follows: ToTalNumberOfTokens=3 FileTokenPosition=1 LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .mod)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation,

these libraries are added to the generated makefile.

Note: If you select Release Build Set, these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default value is
$(OMROOT)\LangC\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT),
$(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is Cleared.

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

(Default = .cc)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The default values are as follows:

Environment Default Value QNXNeutrinoCW $OMROOT/DLLs/CodeWarriorIDE.dll INTEGRITY Empty string IntegrityESTL VxWorks $OMROOT/DLLs/TornadoIDE.dll

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file. The default values are as follows:

(Default = empty string)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch

file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" $makefile $maketarget")

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message.

If you are using a full-featured external code generator, this property setting is ignored.

(Default = $OMROOT/etc/IntegrityMakefileGenerator.bat)

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property

C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .bld)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = work)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (error|catastrophic error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

(Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment. The extension ".h" is the default for C.

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

The default value is Checked:

INTEGRITY5

This metaclass contains the properties that manipulate the INTEGRITY5 operating system environment.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

This property reflects the change in Version 4.1 where the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created.

This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment. The default values are as follows:

Environment Default Value Borland __asm __finallynaked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 thread dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance Microsoft MicrosoftDLL MSStandardLibrary GNAT Empty string INTEGRITY IntegrityESTL JDK Linux MontaVista OsePPCDiab QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks OseSfk receive __asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension -l$(LibPrefix)AomAnim$(BLDTarget)$OMLibSuffix$LibExtension -l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -D_OMINSTRUMENT

BLDAdditionalDefines

The BLDAdditionalDefines property enables you to specify additional compiler preprocessor flags. The default values are as follows:

Environment Default Value INTEGRITY Empty string MultiWin32 IntegrityESTL ESTL

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches. The default values are as follows:

Environment Default Value INTEGRITY :optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 IntegrityESTL :optimizestrategy=space :driver_opts=--diag_suppress=14 :driver_opts=--diag_suppress=550 :cx_mode=extended_embedded :cx_lib=eece :stdcxxincdirs=$(INTEGRITY_ROOT)\eecxx :stdcxxincdirs=$(INTEGRITY_ROOT)\ansi MultiWin32 :cx_template_option=noimplicit :add_output_ext=checked :cx_e_option=msgnumbers :cx_option=exceptions :check=bounds :check=assignbound :check=nilderef :cx_template=local :cx_remark=14 :cx_remark=161 :cx_remark=837 :cx_remark=817 :cx_remark=815 :cx_remark=47 :cx_remark=69 :cx_remark=830 :cx_remark=550 :prelink.args=-r :prelink.args=-X7

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

(Default = empty MultiLine)

BLDMainExecutableOptions

The BLDMainExecutableOptions property specifies the options generated in the main build file of the executable component of the model.

The C default value is as follows:

-G -Ospace -dynamic -non_shared

BLDMainLibraryOptions

The BLDMainLibraryOptions property specifies the options generated in the main build file of the library component of the model.

The C default value is as follows:

-G -Ospace -non_shared

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link. The C default value is "mbx800."

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag

(:cx_option=exceptions).

- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Integrity5Make.bat\" IntegrityBuild.bat buildLibs $BLDTarget "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileDebug

The CompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

There is not default value.

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

The default value is as follows: -D_DEBUG -G .

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

There is not default value.

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

DebugSwitches

The DebugSwitches property sets the debug level used in debug switches. The default values are as follows:

Environment Possible Values Default Value INTEGRITY Default, Multi, None, Plain, and Stack Default OBJECTADA -ga, -gc, -ga -gc -ga RAVEN_PPC -ga, -gc, -ga -gc -ga SPARK Empty string

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

The default value is "main."

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .mod)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note: If you select Release Build Set, these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default value is $(LibPrefix)WebComponents$(BLDTarget)$OMLibSuffix$LibExtension , $(OMRoot)/lib/$(FrameworkLibPrefix)WebServices$(BLDTarget)$OMLibSuffix$LibExtension.

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Cleared)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The default values are as follows:

Environment Default Value QNXNeutrinoCW $OMROOT/DLLs/CodeWarriorIDE.dll INTEGRITY Empty string IntegrityESTL VxWorks $OMROOT/DLLs/TornadoIDE.dll

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment.

(Default = .c)

IntegrityLinkFile

The name of the Integrity link file that should be added to the makefile template.

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file. The default values are as follows:

Environment Default Value Borland "$executable" GNAT Microsoft MicrosoftDLL MSStandardLibrary MultiWin32 (Ada) NucleusPLUS-PPC OBJECTADA RAVEN_PPC SPARK INTEGRITY Empty string IntegrityESTL MicrosoftWinCE.NET MontaVista JDK "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\jdkrun.bat\" $makefile Main$ComponentName" Linux $executable MultiWin32 (C++) QNXNeutrinoCW QNXNeutrinoGCC MicrosoftWinCE "$OMROOT\etc\msceRun.bat" $executable IX86EM OsePPCDiab "$OMROOT/etc/osesfkRun.bat" $executable OseSfk Solaris2 xterm -e $executable Solaris2GNU

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

The InvokeMake default values are as follows:

Environment Default Value Borland $OMROOT/etc/Executer.exe "\"$OMROOT\etc\bc5make.bat\" $makefile $maketarget" GNAT "$makefile" $maketarget INTEGRITY ESTL "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" $makefile $maketarget" Integrity JDK "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\jdkmake.bat\" $makefile $maketarget" Linux $OMROOT/etc/linuxmake $makefile $maketarget Microsoft "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget" MicrosoftDLL MSStandardLibrary MicrosoftWinCE "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\mscemake.bat\" $makefile $maketarget IX86EM" MicrosoftWinCE.NET "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msceNETmake.bat\" $makefile $maketarget x86" MontaVista "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\mvlinuxmake.bat\" $makefile $maketarget" MultiWin32 (Ada) "$OMROOT/etc/Executer.exe" "$OMROOT\etc\AdaMultiWin32Make.bat $makefile $maketarget" OBJECTADA "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaMake.bat $makefile $maketarget" OsePPCDiab "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\oseppcdiabmake.bat\" $makefile $maketarget" OseSfk "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\osesfkmake.bat\" $makefile $maketarget" QNXNeutrinoCW "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\qnxcwmake.bat\" $makefile $maketarget" QNXNeutrinoGCC Empty string RAVEN_PPC "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaRavenPPCMake.bat $makefile $maketarget" Solaris2 $OMROOT/etc/sol2make $makefile $maketarget Solaris2GNU SPARK "$OMROOT/etc/Executer.exe" "$OMROOT\etc\SPARKMake.bat $makefile $maketarget" VxWorks "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vxmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LibPrefix

Combines all of the prefixes of the library names. The C default value is $(FrameworkLibPrefix)$(OMMultipleAddressSpacesPrefix).

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode. The default values are as follows:

Environment Default Value Borland Empty string (blank) GNAT Microsoft MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary MultiWin32 NucleusPLUS-PPC OseSfk INTEGRITY -G IntegrityESTL Linux -g MontaVista OsePPCDiab QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode. The default values are as follows:

Environment Default Value Borland Empty string GNAT INTEGRITY IntegrityESTL Microsoft MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary MultiWin32 NucleusPLUS-PPC OsePPCDiab OseSfk VxWorks Linux -O MontaVista QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

Environment Default Value Borland $OMLinkCommandSet Linux MontaVista MultiWin32 (C++) NucleusPLUS-PPC OsePPCDiab QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks INTEGRITY --one_instantiation_per_object $OMLinkCommandSet -cpu=$(TARGET_CPU) -map IntegrityESTL Microsoft $OMLinkCommandSet /NOLOGO MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary OseSfk -nologo $OMLinkCommandSet QNXNeutrinoCW -static

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .gpj)

MakeFileContentForExe1

The MakeFileContentForExe1 property is the content of the makefile for an executable component type. The default value is as follows:

#!gbuild primaryTarget=$PrimaryTarget [INTEGRITY Application] -bsp $BLDTarget -os_dir $IntegrityRoot -o $OMTargetName$ExeExtension -object_dir=$ObjectsDirectory $BLDMainExecutableOptions $OMMultipleAddressSpacesSwitches $KernelProject $MakeFileNameForExe2 $IntegrateFile $BSPFile $ConnectionFile $ResourceFile $OMMultipleAddressSpacesAdditionalFiles

MakeFileContentForExe2

The MakeFileContentForExe2 property is the content of the makefile for an executable component type. The default value is as follows:

#!gbuild [Program] -o $OMTargetName -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangC -L$OMRoot/LangC/lib $OMUserIncludePath $LinkSwitches $OMCompilationFlag $CompileSwitches $OMReusableFlag $OMInstrumentationFlags $OMInstrumentationLibs $OMMultipleAddressSpacesLibraries $BLDAdditionalDefines $OMUserLibs $OMMainFiles$ImpExtension $OMSrcFiles $IntegrityLinkFile $LinkerFile

MakeFileContentForLib1

The MakeFileContentForLib1 property provides the content of the makefile for a library component type. The default value is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -bsp $BLDTarget -os_dir $IntegrityRoot -object_dir=$ObjectsDirectory $BLDMainLibraryOptions $OMMultipleAddressSpacesSwitches $KernelProject $MakeFileNameForLib2 $IntegrateFile $BSPFile $ConnectionFile $ResourceFile $OMMultipleAddressSpacesAdditionalFiles

MakeFileContentForLib2

The MakeFileContentForLib2 property provides the content of the makefile for a library component type. The default value is as follows:

#!gbuild [Library] -o $OMTargetName$LibExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions -I$OMRoot/LangC $OMUserIncludePath $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $OMReusableFlag $BLDAdditionalDefines $OMSrcFiles

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

The MakeFileNameForExe1 property is the name of the makefile for an executable component type.

The default value is $(OMTargetName)$MakeExtension.

MakeFileNameForExe2

The MakeFileNameForExe2 property is the name of the makefile for an executable component type.

The default value is $(OMTargetName)_program$MakeExtension.

MakeFileNameForLib1

The MakeFileNameForLib1 property is the name of the makefile for a library component type.

The default value is $(OMTargetName)$MakeExtension.

MakeFileNameForLib2

The MakeFileNameForLib2 property is the name of the makefile for a library component type.

The default value is $(OMTargetName)_library$MakeExtension.

MultipleAddressSpacesIntFileContent

The MultipleAddressSpacesIntFileContent property provides the content of the MultipleAddressSpacesIntFileName file with the number of the Integrate configuration file for compiling a list of address spaces that use also POSIX shared memory manager.

The content values are as follows:

Kernel Filename DynamicDownload EndKernel $OMSubComponentInfo AddressSpace shared_memory_manager Filename posix_shm_manager MaximumPriority 200 Language $OMLanguage Task Initial StartIt true EndTask EndAddressSpace

MultipleAddressSpacesIntFileName

The MultipleAddressSpacesIntFileName property identifies a file with this name to be created in case of multiple address space compilation.

The default value is $OMTargetName.int.

MultipleAddressSpacesLibraries

The MultipleAddressSpacesLibraries property names of libraries to add in case of multiple address space usage.

The default value is as follows:

-l$(FrameworkLibPrefix)Dox$(BLDTarget)$LibExtension -llibposix$LibExtension
-llibshm_client$LibExtension

MultipleAddressSpacesPrefix

The MultipleAddressSpacesPrefix property specifies the prefix that is added to libraries in case of multiple address space compilation. OMMultipleAddressSpacesPrefix keyword will add this prefix when needed.

The default value is Distributed.

MultipleAddressSpacesSwitches

A switch for multiple address space compilation.The makefile template can add it directly but it is preferred to use the keyword OMMultipleAddressSpacesSwitches – that checks whether this switch should be added.

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

The default value is "work."

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

The C default value is .o.

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

The default value is Cleared.

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

The C default value is ([^"]+)"[,][ ]line ([0-9]+)[:] (warning|error|catastrophic error).

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (error|catastrophic error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)"[,][ ]line ([0-9]+)[:] (warning)

PosixSharedMemoryFiles

This list is copied only in case of multiple address space compilation.

The default value is $OMROOT/MakeTmpl/posix_shm_manager.gpj,$OMROOT/MakeTmpl/shm_area.c.

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

The default value is Cleared.

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

(Default = empty string)

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

The default value is OM_REUSABLE_STATECHART_IMPLEMENTATION.

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment. The extension ".ads" is the default for Ada.

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default =
-l$OMRoot/LangCpp/lib/$(FrameworkLibPrefix)TomTraceRiC$(BLDTarget)$OMLibSuffix$LibExtension
-l$OMRoot/LangCpp/lib/$(FrameworkLibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension
-l$OMRoot/LangCpp/lib/$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)OxfInstTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMTRACER

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

The default value is Cleared.

WebInstLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on.

Link

The Link metaclass contains a property that controls how links are displayed in object model diagrams.

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the

model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for the element has been applied.

Default = "Default"

Linux

The Linux metaclass contains the Environment settings (Compiler, framework libraries, etc.) for Linux.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/Linux)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

AnimIncludeDirectories

The property AnimIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangC/aom $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangC/lib/linuxaomanim$(LIB_EXT)

AnimOxfLibs

The property AnimOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangC/lib/linuxoxfinst$(LIB_EXT) $(OMROOT)/LangC/lib/linuxomcomappl$(LIB_EXT)

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMANIMATOR

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag

(:cx_option=exceptions).

- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

The default value for MultiWin32 is DebugNoExp; for the other environments, the default value is Debug.

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = $OMROOT/etc/linuxmake linuxbuild.mak build

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompilerFlags

The property CompilerFlags allows you to define additional compilation flags. The value of the property is inserted into the value of the property CompileSwitches (Linux) or CPPCompileSwitches (cygwin). In the generated makefile, you can see the value of this property in the line that begins with ConfigurationCPPCompileSwitches=.

Default = Blank

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

-I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) $OMCPPCompileCommandSet -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default values are as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

The default value is -g.

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

The default value is -O.

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

The default value is "main."

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is "ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2."

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

The C default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note: If you select Release Build Set, these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default value is $(OMROOT)/LangC/lib/linuxWebComponents$(LIB_EXT), $(OMROOT)/lib/linuxWebServices$(LIB_EXT).

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(C Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

The C default value is "include."

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = empty string)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from

the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment. The default values are as follows:

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode. The default values are as follows:

(Default = -g)

LinkerFlags

The property LinkerFlags allows you to define linker flags. The value of the property is inserted into the value of the property LinkSwitches. In the generated makefile, you can see the value of this property in the line that begins with LINK_FLAGS=.

Default = -lpthread -lstdc++

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode. The default values are as follows:

(Default = -O)

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags

stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS=

INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################## Linking instructions ################### ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NoneIncludeDirectories

The property NoneIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = Blank

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = Blank

NoneOxfLibs

The property NoneOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangC/lib/linuxoxf$(LIB_EXT)

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Checked)

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95.

The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

(Default = empty string)

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change. The default values are as follows:

Environment Default Value Borland -DOM_REUSABLE_STATECHART_IMPLEMENTATION Linux NucleusPLUS-PPC OsePPCDiab OseSfk QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks Microsoft /D "OM_REUSABLE_STATECHART_IMPLEMENTATION" MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary INTEGRITY OM_REUSABLE_STATECHART_IMPLEMENTATION IntegrityESTL MontaVista MultiWin32

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

The extension ".h" is the default for C.

TraceIncludeDirectories

The property TraceIncludeDirectories is used to specify the directories that must be referenced in the makefile for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_INCLUDES.

Default = $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/aom $(INCLUDE_QUALIFIER)$(OMROOT)/LangCpp/tom

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = $(OMROOT)/LangCpp/lib/linuxtomtraceRiC$(LIB_EXT) $(OMROOT)/LangCpp/lib/linuxomcomappl$(LIB_EXT) $(OMROOT)/LangCpp/lib/linuxoxf$(LIB_EXT) $(OMROOT)/LangC/lib/linuxaomtrace$(LIB_EXT)

TraceOxfLibs

The property TraceOxfLibs is used to specify the framework libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with OXF_LIBS.

Default = $(OMROOT)/LangC/lib/linuxoxfinst$(LIB_EXT)
$(OMROOT)/LangC/lib/linuxomcomappl$(LIB_EXT)

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = $(DEFINE_QUALIFIER)OMTRACER

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style.

If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

(Default = Checked)

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

The default value is Checked.

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

The default value is Checked.

The Microsoft metaclass contains the Environment settings (Compiler, framework libraries, etc.) for Microsoft compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/WIN32)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = __asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"\"$OMROOT\"\etc\msmake.bat msbuild.mak build \"USE_PDB=FALSE\" \""

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

/I . /I $OMDefaultSpecificationDirectory /I "$(OMROOT)\LangC" /I "$(OMROOT)\LangC\oxf" /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default is $(CREATE_OBJ_DIR) $(CC) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath".

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

The default value is /Zi /Od /D "_DEBUG" /MDd /Fd"$(TARGET_NAME)" .

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

The default value is /Ox /D"NDEBUG" /MD /Fd"$(TARGET_NAME)".

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies .

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

The default value is "main."

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is "ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2."

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set, these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default is $(OMROOT)\LangC\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT), ws2_32$(LIB_EXT).

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = INCLUDE)

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = "$executable")

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $.

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget")

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .lib)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

(Default = empty string)

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet /NOLOGO)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ############### ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables.

Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ############### ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT)

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################# Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = if exist $OMFileObjPath erase $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .obj)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages.

The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

The default is ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error).

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (NMAKE|LINK)(.)(fatal error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = \ )

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95.

The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Cleared)

MicrosoftIDF

The MicrosoftIDF metaclass contains the Environment settings (Compiler, framework libraries, etc.) for MicrosoftIDF compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/WIN32)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = __asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of

arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

/I . /I "$(OMROOT)/LangC" /I "$(OMROOT)/LangC/idf" /I "$(OMROOT)/LangC/idf/Adapters/WIN32" /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default values are as follows:

$(CC) $OMFileCPPCompileSwitches /Fo"$OMFileObjPath" "$OMFileImpPath"

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = /Zi /Od /D "_DEBUG" /MDd /Fd"$(TARGET_NAME)")

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = /Ox /D"NDEBUG" /MD /Fd"$(TARGET_NAME)")

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is "ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2."

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property

ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\cygwinrun.bat\" $executable")

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

The InvokeMake default values are as follows:

Environment Default Value Borland $OMROOT/etc/Executer.exe "\"$OMROOT\etc\bc5make.bat\" $makefile $maketarget" GNAT "$makefile" $maketarget INTEGRITY ESTL "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\IntegrityMake.bat\" $makefile $maketarget" Integrity JDK "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\jdkmake.bat\" $makefile $maketarget" Linux $OMROOT/etc/linuxmake $makefile $maketarget Microsoft "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msmake.bat\" $makefile $maketarget" MicrosoftDLL MSStandardLibrary MicrosoftWinCE "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\mscemake.bat\" $makefile $maketarget IX86EM" MicrosoftWinCE.NET "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\msceNETmake.bat\" $makefile $maketarget x86" MontaVista "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\mvlinuxmake.bat\" $makefile $maketarget" MultiWin32 (Ada) "$OMROOT/etc/Executer.exe" "$OMROOT\etc\AdaMultiWin32Make.bat $makefile $maketarget" OBJECTADA "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaMake.bat $makefile $maketarget" OsePPCDiab "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\oseppcdiabmake.bat\" $makefile $maketarget" OseSfk "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\osesfkmake.bat\" $makefile $maketarget" QNXNeutrinoCW "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\qnxcwmake.bat\" $makefile $maketarget" QNXNeutrinoGCC Empty string RAVEN_PPC "$OMROOT/etc/Executer.exe" "$OMROOT\etc\ObjectAdaRavenPPCMake.bat $makefile $maketarget" Solaris2 $OMROOT/etc/sol2make $makefile $maketarget Solaris2GNU SPARK "$OMROOT/etc/Executer.exe" "$OMROOT\etc\SPARKMake.bat $makefile $maketarget" VxWorks "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vxmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

Environment Default Value Borland $OMLinkCommandSet Linux MontaVista MultiWin32 (C++) NucleusPLUS-PPC OsePPCDiab QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks INTEGRITY --one_instantiation_per_object $OMLinkCommandSet -cpu=$(TARGET_CPU) -map IntegrityESTL Microsoft $OMLinkCommandSet /NOLOGO MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary OseSfk -nologo $OMLinkCommandSet QNXNeutrinoCW -static

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll

ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################## Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build. The default value is as follows:

if exist $OMFileObjPath erase $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile. The default values are as follows:

Environment Default Value INTEGRITY work Integrity ESTL MultiWin32 obj_dir All others Empty string

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .obj)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages.

The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (NMAKE)(.)(fatal error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = [:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

ModelElement

The metaclass ModelElement contains properties that can be used to customize code generation by changing the way that Rational Rhapsody handles specific elements when it transforms a model into a simplified model before generating code.

In general, the properties in this metaclass relate to model elements that can be found under other types of model elements, for example, descriptions and annotations. These properties are therefore visible at different project levels - for example, package, class, and attribute.

SimplifyAnnotations

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyAnnotations can be used to change the way annotations are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Annotations is ignored.
- Copy - Annotations will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for Annotations, as defined in Rational Rhapsody.

- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for Annotations has been applied.

Default = "Default"

SimplifyDescription

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyDescription can be used to change the way Descriptions are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Descriptions is ignored.
- Copy - Descriptions will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for Descriptions, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for Descriptions has been applied.

Default = "Default"

SimplifyExternal

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyExternal can be used to change the way that code is generated for external elements that are not actually part of the model, for example, base classes for classes in the model, by changing the way that such elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The elements are ignored.
- Copy - The elements will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for these elements, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for these element has been applied.

Default = "Default"

SimplifyInstrumentation

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyInstrumentation can be used to customize the generation of instrumentation code (such as animation) by changing the way instrumentation is handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Instrumentation is ignored.
- Copy - Instrumentation will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for instrumentation, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for instrumentation has been applied.

Default = "Default"

SimplifyProperties

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyProperties can be used to customize the way that overridden properties affect code generation by changing the way that Rational Rhapsody handles these properties when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Overridden properties is ignored.
- Copy - Overridden properties will just be copied from the original to the simplified model. Their effect will not be modified in any way.
- Default - Uses the standard simplification for overridden properties, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for overridden properties has been applied.

Default = "Default"

SimplifyStandardOperations

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyStandardOperations can be used to customize the way that code is generated for operations defined using the "StandardOperation" properties by changing the way that Rational Rhapsody handles such operations when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Standard Operations is ignored.

- Copy - Standard Operations will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for Standard Operations, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for Standard Operations has been applied.

Default = "Default"

SimplifyWebify

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyWebify can be used to customize the generation of Webify code by changing the way Webify is handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Webify is ignored.
- Copy - Webify will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for Webify, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for Webify has been applied.

Default = "Default"

Multi4Win32

The Multi4Win32 metaclass contains theEnvironment settings (Compiler, framework libraries, etc.) for Multi4Win32 compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

This property reflects the change in Version 4.1 where the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created.

This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

AnimInstLibs

The property AnimInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension -l$(LibPrefix)AomAnim$(BLDTarget)$OMLibSuffix$LibExtension -l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension

AnimPreprocessor

The property AnimPreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Animation. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -D_OMINSTRUMENT

BLDAdditionalOptions

The BLDAdditionalOptions property enables you to specify additional compilation switches.

Default = -I. -threading=multiple --exceptions --no_implicit_include --display_error_number --diag_remark 14,161,837,817,815,47,69,830,550 -prelink.args=-r -prelink.args=-X7 -language=cxx

BLDIncludeAdditionalBLD

The BLDIncludeAdditionalBLD enables you to specify additional build options.

(Default = empty MultiLine)

BLDTarget

The BLDTarget property specifies the target BSP. For example, ":target=Win32". This property also affects the names of the framework libraries used in the link.

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of

arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

The default value for MultiWin32 is DebugNoExp; for the other environments, the default value is Debug.

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\MultiWin32Make.bat\" MultiWin32Build.bat

buildLibs "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

COM

The COM property specifies whether the current component is a COM component.

By default, this property is set to Checked for all COM components (stereotypes COM DLL, COM EXE, and COM TLB).

If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows.

(Default = Cleared)

CompileDebug

The CompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

CompileRelease

The CompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property. The default values are as follows:

Environment Default Compile Switches Borland -I$OMDefaultSpecificationDirectory -I$(BCROOT)\INCLUDE;.;"$(OMROOT)\LangCpp"; "$(OMROOT)\LangCpp\oxf";"$(OMROOT)\LangCpp\omCom"; -D_RTLDLL;_AFXDLL;WIN32;_CONSOLE;_MBCS; WINDOWS;BORLAND;_BOOLEAN $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) $OMCPPCompileCommandSet -c Linux MontaVista -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp

-I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c Microsoft MicrosoftDLL /I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MicrosoftWinCE /I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX- /D _WIN32_WCE=$(CEVersion) /D "$(CEConfigName)" $(MACHINE_C_FLAGS) /D "_OM_NO_IOSTREAM" /D UNDER_CE=$(CEVersion) /D "UNICODE" /D "_OM_UNICODE_ONLY" $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MicrosoftWinCE.NET /I . /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX- /D _WIN32_WCE=$(CEVersion) /D "$(CEConfigName)" $(MACHINE_C_FLAGS) /D "_OM_NO_IOSTREAM" /D UNDER_CE=$(CEVersion) /D "UNICODE" /D "_OM_UNICODE_ONLY" $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /D "_X86_" /c MSStandardLibrary /I . /I $OMDefaultSpecificationDirectory /I $(OMROOT)\LangCpp /I $(OMROOT)\LangCpp\oxf /nologo /W3 /GX $OMCPPCompileCommandSet /D "_AFXDLL" /D "WIN32" /D "_CONSOLE" /D "_MBCS" /D "_WINDOWS" /D "OM_USE_STL" $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) /c MultiWin32 ${CPPCompileDebugNoExp} $CPPAdditionalCompileSwitches Nucleus PLUS-PPC -v -c -DPLUS -DUSE_IOSTREAM -D__DIAB -t$(CPU) -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)\LangCpp -I$(OMROOT)\LangCpp\oxf -I$(ATI_DIR) -Xmismatch-warning -Xno-common $OMCPPCompileCommandSet $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) OsePPCDiab OseSfk -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)$/LangCpp $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) QNXNeutrinoCW -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c QNXNeutrinoGCC Solaris2 Solaris2GNU -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DUSE_IOSTREAM $OMCPPCompileCommandSet -c VxWorks -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangCpp -I$(OMROOT)/LangCpp/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -c

CPPCompileSwitches

The CPPCompileSwitches property specifies the compiler switches. The default value is as follows:

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component. The default value is as follows:

-D_DEBUG -G

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

There is no default value.

DebugLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or ReleaseLibSuffix according to the compilation to the build type: Release/Debug.

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

The default value is "main."

See also the definition of the EntryPointDeclarationModifier property for more information.

EnvironmentVarName

The EnvironmentVarName property specifies the name of the global variable that you must define in order to use the compiler. It is used by the MultiMakefileGenerator. The value replaces the "$value of the EnvironmentVarName" keyword inside the property value BLDAdditionalOptions.

(Default = MULTI_ROOT)

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property.

ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression

- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .exe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

ExtraFilesToCopy

A list of file names (with full paths) separated with commas. The generator copies this list of files to the folder of the makefile only if the file does not already exist.

FrameworkLibPrefix

The FrameworkLibPrefix property specifies the prefix of the Rational Rhapsody framework library linked with your application.

(Default = Multi4Win32)

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled).

During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set, these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

The default value is as follows:

$(LibPrefix)WebComponents$(BLDTarget)$OMLibSuffix$LibExtension , $(OMRoot)/lib/$(FrameworkLibPrefix)WebServices$(BLDTarget)$OMLibSuffix$LibExtension

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

InitTracingCppSupport

The InitTracingCppSupport property specifies

(Default = _main();)

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = $executable)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

The InvokeMake default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\Multi4Win32Make.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation.

If the specified path is incorrect, Rational Rhapsody generates an error message.If you are using a full-featured external code generator, this property setting is ignored.

(Default = $OMROOT/etc/MultiMakefileGenerator.exe)

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .lib)

LibPrefix

Combines all the prefixes of library names.

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

There is no default.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

There is no default.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

There is no default.

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .gpj)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags

stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables.

Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF

SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################# Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileContentForExe1

This property is the content of the makefiles, in the case of an executable component type.

The default value is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -object_dir=$ObjectsDirectory $MakeFileNameForExe2

MakeFileContentForExe2

This property is the content of the makefiles, in the case of an executable component type.

The default value is as follows:

#!gbuild [Program] -o $OMTargetName$ExeExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions $BLDIncludeAdditionalBLD -I$(OMRoot)/LangC -L$(OMRoot)/LangC/lib $OMUserIncludePath $LinkSwitches $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $OMInstrumentationLibs $BLDAdditionalDefines $OMUserLibs $OMMainFiles$ImpExtension $OMSrcFiles

MakeFileContentForLib1

This property is the content of the makefiles, in the case of a library component type.

The default value is as follows:

#!gbuild primaryTarget=$PrimaryTarget [Project] -object_dir=$ObjectsDirectory $MakeFileNameForLib2

MakeFileContentForLib2

This property is the content of the makefiles, in case of library component type.

The default value is as follows:

#!gbuild [Library] -o $OMTargetName$LibExtension -object_dir=$ObjectsDirectory $BLDAdditionalOptions $BLDIncludeAdditionalBLD -I$(OMRoot)/LangC $OMUserIncludePath $OMCompilationFlag $CompileSwitches $OMInstrumentationFlags $BLDAdditionalDefines $OMSrcFiles

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

MakeFileNameForExe1

This property contains the name of the makefiles, in the case of an executable component type.

The default value is as follows:

$(OMTargetName)$MakeExtension

MakeFileNameForExe2

This property contains the name of the makefiles, in the case of an executable component type.

The default value is as follows:

$(OMTargetName)_program$MakeExtension

MakeFileNameForLib1

This property contains the name of the makefiles, in the case of a library component type.

The default value is as follows:

$(OMTargetName)$MakeExtension

MakeFileNameForLib2

The name of the makefiles, in case of library component type.

The default value is as follows:

$(OMTargetName)_library$MakeExtension

MultipleAddressSpacesIntFileContent

The content of the MultipleAddressSpacesIntFileName file.

MultipleAddressSpacesIntFileName

The MultipleAddressSpacesIntFileName property provides a file with this name is created in case of multiple address space compilation.

MultipleAddressSpacesLibraries

The MultipleAddressSpacesLibraries property provides the names of libraries to add in case of multiple address space usage.

MultipleAddressSpacesPrefix

A prefix that is added to libraries in case of multiple address space compilation. OMMultipleAddressSpacesPrefix keyword adds this prefix when needed.

MultipleAddressSpacesSwitches

A switch for multiple address space compilation.The makefile template can add it directly but it is preferred to use the keyword OMMultipleAddressSpacesSwitches – that checks whether this switch should be added.

NetAndSocketLibs

A list of library names that is added to OMWebLibs keyword if web-enabling flag is on or to OMInstrumentationFlags keyword if the instrumentation is in animation mode.

NoneInstLibs

The property NoneInstLibs is used to specify the static libraries required when Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension

NonePreprocessor

The property NonePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to None. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = Blank

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build. The default values are as follows:

Environment Clean Command Borland if exist $OMFileObjPath erase $OMFileObjPath INTEGRITY IntegrityESTL Microsoft MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary GNAT Empty string MultiWin32 JDK if exist $OMFileObjPath del $OMFileObjPath Linux $(RM) $OMFileObjPath MontaVista OsePPCDiab OseSfk QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks NucleusPLUS-PPC @if exist $OMFileObjPath $(RM) $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = work)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment. The default values are as follows:

(Default = .obj)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive. The default values are as follows:

Environment Default Value Borland Cleared GNAT INTEGRITY IntegrityESTL JDK Microsoft MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary MultiWin32 NucleusPLUS-PPC OseSfk OsePPCDiab RAVEN_PPC SPARK VxWorks Linux Checked MontaVista QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (error|warning) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

The default values are as follows:

Environment Default Value Borland PsosX86 ToTalNumberOfTokens=5,FileTokenPosition=4,LineTokenPosition=5 GNAT ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2 JDK Linux MontaVista QNXNeutrinoCW QNXNeutrinoGCC Solaris2GNU SPARK VxWorks IntegrityESTL

ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2 Microsoft MicrosoftDLL MicrosoftWinCE.NET MSStandardLibrary NucleusPLUS-PPC OsePPCDiab OseSfk Solaris2

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (Error)[:] (build failed)

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^"]+)", line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

The default value is Checked.

RCCompileCommand

The RCCompileCommand property is a string that specifies the compilation command for the resource file.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = empty string)

RCExtension

The RCExtension property is a string that specifies the extension for resource files.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = .rc)

ReleaseLibSuffix

A suffix added to library names. OMLibSuffix keyword is replaced with this property or DebugLibSuffix according to the compilation to the build type: Release/Debug.

(Default = R)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely.

In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to Checked. If UseRemoteHost is Checked and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

(Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker.

The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

(Default = /SUBSYSTEM:console)

TraceInstLibs

The property TraceInstLibs is used to specify the static libraries required when Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_LIBS.

Default = -l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)AomTrace$(BLDTarget)$OMLibSuffix$LibExtension
-l$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension
-l$(LibPrefix)OxfInst$(BLDTarget)$OMLibSuffix$LibExtension
-l$OMRoot/LangCpp/lib/$(FrameworkLibPrefix)TomTraceRiC$(BLDTarget)$OMLibSuffix$LibExtension
-l$OMRoot/LangCpp/lib/$(FrameworkLibPrefix)Oxf$(BLDTarget)$OMLibSuffix$LibExtension
-l$OMRoot/LangCpp/lib/$(FrameworkLibPrefix)OmComAppl$(BLDTarget)$OMLibSuffix$LibExtension
-lwsock32.lib

TracePreprocessor

The property TracePreprocessor is used to specify conditions that should be used for conditional compilation for projects where Instrumentation Mode is set to Tracing. In the makefile, these will appear in the line that begins with INST_FLAGS.

Default = -DOMTRACER -DRIC_APP

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors checkmark (located in the configuration Initialization tab).

(Default = Cleared)

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

The default value for the following environments is Cleared:

Borland GNAT INTEGRITY IntegrityESTL Microsoft MicrosoftDLL MSStandardLibrary MultiWin32

The default value for the following environments is Checked:

Linux MicrosoftWinCE.NET MontaVista NucleusPLUS-PPC OsePPCDiab OseSfk QNXNeutrinoCW QNXNeutrinoGCC Solaris2 Solaris2GNU VxWorks

NucleusPLUS-PPC

The NucleusPLUS-PPC metaclass contains the Environment settings (Compiler, framework libraries, etc.) for NucleusPLUS-PPC compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.

- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/Nucleus)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\numake.bat" nubuild.mak buildLibs \"CPU=$CPU\" \"BUILD_SET=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

-v -c -DUSE_STDIO -DPLUS -t$(CPU) -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -I$(ATI_DIR) -I$(ATI_DIR)/plus -Xoptimized-debug-on -XO -Xsize-opt -Xmismatch-warning -Xno-common $OMCPPCompileCommandSet $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES)

CPPAdditionalReservedWords

The CPPAdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

The default value is as follows:

__asm __finally naked __based __inline __single_inheritance __cdecl __int8 __stdcall __declspec __int16 dllexport __int32 __try dllimport __int64 __virtual_inheritance __except __leave __fastcall __multiple_inheritance

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default values are as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

The default value is -g -D_DEBUG -DASSERT_DEBUG.

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

The default value is -DNDDEBUG.

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $(CREATE_OBJ_DIR) $OMFileDependencies.

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

The default value is "numain."

See also the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is "ToTalNumberOfTokens=3,FileTokenPosition=1,LineTokenPosition=2."

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .elf)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

The C default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled).

During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries). The default values are as follows:

The default value is as follows:

$(OMROOT)\LangC\lib\$(LIB_PREFIX)WebComponents$(LIB_POSTFIX)$(LIB_EXT), $(OMROOT)\lib\$(LIB_PREFIX)WebServices$(LIB_POSTFIX)$(LIB_EXT)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(C Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = .INCLUDE:)

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = "$executable")

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch

file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $.

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\numake.bat\" $makefile $maketarget")

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format.

If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .lib)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

(Default = empty string)

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################## Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib

LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches
LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ################ Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################## Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows:

########## Generated dependencies ##################
$OMContextDependencies
$OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches)
/Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################## Linking instructions ##################
##
$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath
$OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD)
$OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \
$(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT)
$(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo
Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS)
$(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase
$OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase
$(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase
$(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist
$(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = @if exist $OMFileObjPath $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error) (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

The default is ([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error).

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of

compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (error|fatal error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^(]+)[(]([0-9]+)[)] [:] (warning)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Cleared)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95.

The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

Operation

The Operation metaclass contains properties that control operations.

ActivityReferenceToAttributes

The ActivityReferenceToAttributes property specifies whether Rational Rhapsody should generate references in the functor object, thereby giving you direct access to the attributes of the class that owns the modeled operation (without the need for this_). See the section on activity diagrams in the Rational Rhapsody Help for detailed information about modeled operations and functor classes.

(Default = Checked)

AnimAllowInvocation

The AnimAllowInvocation property specifies whether primitive and triggered operations can be called during instrumentation. If an operation is called during animation, its return value is displayed in the output window; if it is traced, the return value is displayed in the console. The possible values are as follows:

- All - Enable all operation calls, regardless of visibility.
- None - Do not enable operation calls.
- Public - Enable calls to public operations only.

- Protected - Enable calls to protected operations only.

(Default = None)

AnimateTriggeredOperationReturnValue

The property AnimateTriggeredOperationReturnValue allows you to specify that the return values of triggered operations should be displayed in animated sequence diagrams.

Default = Checked

DeclarationModifier

The property DeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in operation declarations. Keywords that appear between the return type and the operation name are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and PostDeclarationModifier.

Default = Blank

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

EntryCondition

The EntryCondition property specifies the task guard. (Default = empty string)

GenerateImplementation

The GenerateImplementation property specifies whether to generate the body for the operation.

To generate Import pragmas, add the "pragma..." declaration in the C_CG::Operation::SpecificationEpilog property.

(Default = Checked)

ImplementActivity Diagram

The ImplementActivity Diagram property enables or disables code generation for activity diagrams. (Default = Cleared)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementFlowchart

ImplementFlowchart is a boolean property that specifies whether or not code should be generated for the flow charts created by the user. It can be set at the individual operation level or at higher levels, such as class or package.

Default = Checked

ImplementationName

The ImplementationName property enables you to give an operation one model name and generate it with another name. It is introduced as a workaround that enables you to generate const and non-const operations with the same name. For example:

- Create a class A.
- Add a non-const operation f().
- Add a const operation f_const().
- Set the C_CG::Operation::ImplementationName property for f_const() to “f.”
- Generate the code.

The resulting code is as follows: class A { ... void f(); / the non const f */ ... void f() const; /* actually f_const() */ ... }; The creation of two operations with the same signature, differing only in whether it is a const, is a common practice in C++, especially for STL users. (Default = empty string)*

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside

Package Yes Outside

(Default = empty MultiLine)

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for C You can inline attribute and relation accessors and mutators for increased code performance. The visibility of the inlined operations can be either public or private. The accessor can contain only a return statement, or more than just a return statement. For Rational Rhapsody Developer for C, there are two possible settings for this property:

- none - The operation is not generated inline. For example: /* Mutator of Tank::ItsDishwasher relation */ void Tank_setItsDishwasher(struct Tank_t* const me, struct Dishwasher_t* p_Dishwasher) { if(p_Dishwasher != NULL) Dishwasher__setItsTank(p_Dishwasher, me); Tank__setItsDishwasher(me, p_Dishwasher); } /* Accessor to Tank::ItsDishwasher relation */ struct Dishwasher_t* Tank_getItsDishwasher(const struct Tank_t* const me) { return (struct Dishwasher_t*)me-itsDishwasher; }
- in_header - The operation is generated inline, as follows:
- Mutators are defined as macro definitions. For example: /* Inline Mutator of Tank::ItsDishwasher relation */ #define Tank_setItsDishwasher(me, p_Dishwasher) \ { \ if((p_Dishwasher) != NULL) \ Dishwasher__setItsTank((p_Dishwasher), (me)); \ Tank__setItsDishwasher((me), (p_Dishwasher)); \ }
- Accessors that contain only return statements are defined as macros (the return statement and the semicolon at the end of expression are omitted); other accessors are generated as operations. For example: /* Inline Accessor to Tank::ItsDishwasher relation */ #define Tank_getItsDishwasher(me) ((me)-itsDishwasher)
- If the attribute visibility is defined as Private, the macro definitions are placed in the implementation (.c) file.
- If the attribute visibility is defined as Public, the macro definitions are placed in the specification (.h) file.

Note the following:

- Each instance of the macro’s parameters is parenthesized.
- You cannot inline an accessor if it contains statements other than the return statement. For example, accessors to relations implemented using RiCCollection cannot be generated as function-like macros.
- You cannot set the Inline property separately for specific helpers (for example, only mutators) - this property affects all helpers of the attribute or relation.
- If a multi-lined mutator macro is called as the body of the “then” part of an “if ...else” statement, you must enclose it in parentheses or it generates a compilation error. For example: // Erroneous code: If (itsDishwasher != NULL) Tank_setItsDishwasher(me, itsDishwasher); Else Return; // Correct code: If (itsDishwasher != NULL) { Tank_setItsDishwasher(me, itsDishwasher); } Else Return;

IsAnimationHelper

The IsAnimationHelper property indicates whether the operation should be generated only when animating the model. (Default = Cleared)

IsEntry

The IsEntry property indicates whether the operation is a task entry or a regular operation in AdaTask and AdaTaskType classes. (Default = Cleared)

IsExplicit

The boolean property IsExplicit allows you to specify that a constructor is an explicit constructor. (Default = Cleared)

IsNative

The IsNative property specifies whether the Java modifier "native" should be added to an operation in the source file. The body of such operations, if specified, is ignored by the code generator. (Default = Cleared)

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations.

This property affects class operations, in addition to accessors and mutators for relations and attributes.

The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

(Default = common)

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation. (Default = empty string)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = Auto)

Me

The Me property specifies the name of the first argument to operations generated in C. (Default = me)

MeDeclType

The MeDeclType property is a string that specifies the type of the first argument to operations generated in C, as a pointer to an object or object type. The default value is as follows: $objectName* const The variable $objectName is replaced with the name of the object or object type.

PostDeclarationModifier

The property PostDeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in operation declarations. Keywords that appear after the operation argument list are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties PreDeclarationModifier and DeclarationModifier.

Default = Blank

PreDeclarationModifier

The property PreDeclarationModifier is used to allow Rational Rhapsody to reverse engineer non-standard keywords that appear in operation declarations. Keywords that appear before the return type are stored as the value of this property, and the property is then used during code generation to recreate the original code.

Since this is a code generation property, it can also be used to add non-standard keywords to code even when reverse engineering is not used.

This property is used in conjunction with the properties DeclarationModifier and PostDeclarationModifier.

Default = Blank

PrivateQualifier

The PrivateQualifier property specifies the qualifier that is printed at the beginning of a private operation declaration or definition. You can set this property to an empty string to prevent the generation of the static qualifier in the private function declaration or definition. (Default = static)

ProtectedName

The ProtectedName property specifies the pattern used to generate names of private operations in C. The default value is as follows: $opName

The $opName variable specifies the name of the operation. For example, the generated name of a private operation go() of an object A is generated as: go()

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C. The default value is as follows: $objectName_$opName

The $objectName variable specifies the name of the object; the $opName variable specifies the name of the operation. For example, the generated name of a public operation go() of an object A is generated as: A_go()

PublicQualifier

The PublicQualifier property specifies the qualifier that is printed at the beginning of a public operation declaration or definition. Note that the Static checkmark in the operation dialog UI is disabled in Rational Rhapsody Developer for C because the checkmark is associated with class-wide semantics that are not

supported by Rational Rhapsody Developer for C.

When loading models from previous versions, the Static checkmark is cleared; if the operation is public, the C_CG::Operation::PublicQualifier property value is set to Static in order to generate the same code. (Default = empty string)

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

RenamesKind

The RenamesKind property specifies whether the renaming of the operation designated in the C_CG::Operation::Renames property is “as specification” or “as body.”(Default = Specification)

ReturnTypeByAccess

The ReturnTypeByAccess property determines whether the return type is generated as an access type or a regular type. Note that this property is applicable only to classes for which an access type is generated. (Default = Cleared)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for the element has been applied.

Default = "Default"

SimplifyTriggeredOperation

If you are using the Rational Rhapsody customizable code generation mechanism, the property SimplifyTriggeredOperation can be used to change the way triggered operations are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - Triggered operations is ignored.
- Copy - Triggered operations will just be copied from the original to the simplified model. They will not be modified in any way.
- Default - Uses the standard simplification for triggered operations, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for triggered operations has been applied.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

TaskDefaultScheme

The TaskDefaultScheme property sets the task default entry scheme. The possible values are as follows:

- Conditional
- Timed
- None

(Default = None)

TaskDefaultSchemeDelayStatement

The TaskDefaultScheme property sets the task default entry statement for timed entry schemes. (Default = empty MultiLine)

ThisByAccess

The ThisByAccess property specifies whether to pass the this parameter as an access mode parameter for a non-static operation. (Default = Cleared)

ThisName

The ThisName property enables you to specify the name of the this parameter, which specifies the instance. (Default = this)

ThrowExceptions

The ThrowExceptions property specifies the exceptions that an operation can throw. Separate multiple exceptions with commas. (Default = empty string)

VirtualMethodGenerationScheme

The VirtualMethodGenerationScheme property enables backward-compatibility mode for methods of interface and abstract classes. The possible values are as follows:

- Default - The class type is class-wide, but the this parameters are not.
- ClassWideOperations - The class type is not class-wide, but the this parameters are.

(Default = Default)

Package

The Package metaclass contains properties that affect packages.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

ContributesToNamespace

The ContributesToNamespace property specifies whether the packages contained in this package is declared as children packages of this package. Regardless of the setting, a directory is created for the current package to hold its contained elements. (Default = Checked)

DefineNameSpace

The DefineNameSpace property specifies whether a package defines a namespace. A namespace is a declarative region that attaches an additional identifier to any names declared inside it. (Default = Cleared)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument's description Operation Primitive operations, triggered operations, $Arguments - The operation argument's description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

EventsBaseID

The EventsBaseID property specifies the base ID for events. The default values are as follows:

- For Ada, C, and C++, the default event base ID is –1.
- For Java, the default event base ID is 16.

GenerateDirectory

The GenerateDirectory property specifies whether to generate a separate directory for the package.

The possible values are as follows:

- Checked - The package generates a directory.
- Cleared - The package will not generate a directory. (This is the default.)

GenerateDirectory has an immediate effect on directory generation.

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces. (Default = empty string)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body. (Default = empty MultiLine)

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body. (Default = empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement.

For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = empty MultiLine)

InitializationCode

The InitializationCode property specifies the user-defined initialization code to add to the package body. (empty MultiLine)

IsNested

The IsNested property specifies whether to generate the class or package as nested. (Default = Cleared)

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private. (Default = Cleared)

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package. (Default = Public)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation.

Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before

the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost.

(Default = Auto)

PackageClassNamePolicy

The PackageClassNamePolicy property specifies the naming policy for classes generated by Rhapsody. The possible values are as follows:

- Default - Use the default naming style (the package class name is the same as the package name).
- WithSuffix - Add a suffix to the class name. The suffix is "_pkgClass".

(Default = Default)

PackageEventIdRange

The PackageEventIdRange property specifies the maximum number of events allowed in a package. This property is set on the component level. (Default = 200)

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.

- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for the element has been applied.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification. (Default = empty MultiLine)

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification. (Default = empty MultiLine)

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces. (Default = empty string)

Port

The Port metaclass controls whether code is generated for ports.

Generate

The Generate property specifies whether to generate code for a particular type of element.

(Default = Checked)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody’s standard simplification for the element has been applied.

Default = "Default"

Relation

The Relation metaclass contains properties that affect relations.

Add

The Add property specifies the command used to add an item to a container.

(Default = Add_$target:c)

AddGenerate

The AddGenerate property specifies whether to generate an Add() operation for relations.

(Default = Checked)

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

Clear

The Clear property specifies the name of an operation that removes all items from a relation.

(Default = Clear_$target:c)

ClearGenerate

The ClearGenerate property specifies whether to generate a Clear() operation for relations. (Default = Checked)

CreateComponent

The CreateComponent property specifies the name of an operation that creates a new component in a composite class. (Default = New_$target:c)

CreateComponentGenerate

The CreateComponentGenerate property specifies whether to generate a CreateComponent operation for composite objects. Setting this property to False is one way to optimize your code for size. (Default = Checked)

DataMemberVisibility

The DataMemberVisibility property specifies the visibility of the relation data member. For example, if the relation is implemented as a pointer, this property determines whether the pointer data member is declared as public, private, or protected. The default value for Ada and C is Private; the default value for C++ and Java is Protected.

DeleteComponent

The DeleteComponent property specifies the name of an operation that deletes a component from a composite class. (Default = Delete_$target:c)

DeleteComponentGenerate

The DeleteComponentGenerate property specifies whether to generate a DeleteComponent() operation for composite objects. (Default = Checked)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

Keyword names can be written in parentheses. For example:

$(Name)

If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

Find

The Find property specifies the name of an operation that locates an item among relational objects. (Default = Find_$target:c)

FindGenerate

The FindGenerate property specifies whether to generate a Find() operation for relations.

(Default = Cleared)

Get

The Get property specifies the name of an operation that retrieves the relation currently pointed to by the iterator. (Default = Get_$target:c)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index. The ContainerTypes>::Relationtype::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

(Default = get$cname:cAt)

GetAtGenerate

The GetAtGenerate property specifies whether to generate a getAt() operation for relations. The possible values are as follows:

- Checked - Generate a getAt() operation for relations.
- Cleared - Do not generate a getAt() operation for relations. Setting the GetAtGenerate property to False is one way to optimize your code for size.

(Default = Cleared)

GetEnd

The GetEnd property specifies the name of an operation that points the iterator to the last item in a collection. (Default = Get_$target:cEnd)

GetEndGenerate

The GetEndGenerate property specifies whether to generate a GetEnd() operation for relations. (Default = Checked)

GetGenerate

The GetGenerate property specifies whether to generate accessor operations for relations. (Default = Checked)

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key. For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

(Default = get$cname:c_Key)

GetKeyGenerate

The GetKeyGenerate property specifies whether to generate getKey() operations for relations. Setting this property to False is one way to optimize your code for size.

(Default = Checked)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.

- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = empty MultiLine)

ImplementWithStaticArray

The ImplementWithStaticArray property specifies whether to implement relations as static arrays. The possible values are as follows:

- Default - Rational Rhapsody provides the appropriate implementation for all fixed and bounded relations.
- FixedAndBounded - All fixed and bounded relations are generated into static arrays.

(Default = FixedAndBounded)

InitializeComposition

The InitializeComposition property controls how a composition relation is initialized. The possible values are as follows:

- InInitializer
- InRecordType
- None

(Default = InInitializer)

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for C Beginning with Rational Rhapsody Developer for C Version 4.2, you can inline attribute and relation accessors and mutators for increased code performance. The visibility of the inlined operations can be either public or private. The accessor can contain only a return statement, or more than just a return statement. For Rational Rhapsody Developer for C, there are two possible settings for this property:

- none - The operation is not generated inline. For example: /* Mutator of Tank::ItsDishwasher relation */ void Tank_setItsDishwasher(struct Tank_t* const me, struct Dishwasher_t* p_Dishwasher) { if(p_Dishwasher != NULL) Dishwasher__setItsTank(p_Dishwasher, me); Tank__setItsDishwasher(me, p_Dishwasher); } /* Accessor to Tank::ItsDishwasher relation */ struct Dishwasher_t* Tank_getItsDishwasher(const struct Tank_t* const me) { return (struct Dishwasher_t*)me-itsDishwasher; }
- in_header - The operation is generated inline, as follows:
- Mutators are defined as macro definitions. For example: /* Inline Mutator of Tank::ItsDishwasher relation */ #define Tank_setItsDishwasher(me, p_Dishwasher) \ { \ if((p_Dishwasher) != NULL) \ Dishwasher__setItsTank((p_Dishwasher), (me)); \ Tank__setItsDishwasher((me), (p_Dishwasher)); \ }
- Accessors that contain only return statements are defined as macros (the return statement and the semicolon at the end of expression are omitted); other accessors are generated as operations. For example: /* Inline Accessor to Tank::ItsDishwasher relation */ #define Tank_getItsDishwasher(me) ((me)-itsDishwasher)
- If the attribute visibility is defined as Private, the macro definitions are placed in the implementation (.c) file.
- If the attribute visibility is defined as Public, the macro definitions are placed in the specification (.h) file.

Note the following:

- Each instance of the macro’s parameters is parenthesized.
- You cannot inline an accessor if it contains statements other than the return statement. For example, accessors to relations implemented using RiCCollection cannot be generated as function-like macros.
- You cannot set the Inline property separately for specific helpers (for example, only mutators) - this property affects all helpers of the attribute or relation.
- If a multi-lined mutator macro is called as the body of the “then” part of an “if ...else” statement, you must enclose it in parentheses or it generates a compilation error. For example: // Erroneous code: If (itsDishwasher != NULL) Tank_setItsDishwasher(me, itsDishwasher); Else Return; // Correct code: If (itsDishwasher != NULL) { Tank_setItsDishwasher(me, itsDishwasher); } Else Return;

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased. (Default = Cleared)

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

(Default = common)

ObjectInitialization

The ObjectInitialization property specifies what kind of initialization will occur for the initial instances of a configuration. The possible values are as follows:

- Full - Instances are initialized and their behavior is started.
- Creation - Instances are initialized but their behavior is not started.
- None - Instances are not initialized and their behavior is not started.

(Default = Full)

Remove

The Remove property specifies the name of an operation that removes an item from a relation. (Default = Remove_$target:c)

RemoveGenerate

The RemoveGenerate property specifies whether to generate a Remove() operation for relations. (Default = Checked)

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation,

passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

(Default = remove$cname:c_Key)

RemoveKeyGenerate

The RemoveKeyGenerate property specifies whether to generate a removeKey() operation for qualified relations. Setting this property to False is one way to optimize your code for size. (Default = Checked)

RemoveKeyHelpersGenerate

The RemoveKeyHelpersGenerate property enables you to control the generation of the relation helper methods (for example, _removeItsX() and __removeItsX()). The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the C_CG::Relation::RemoveKey property.

(Default = Checked)

SafeInitScalar

The SafeInitScalar property specifies whether to initialize scalar relations as null pointers. (Default = Cleared)

Set

The Set property specifies the name of the mutator generated for scalar relations. (Default = Set_$target:c)

SetGenerate

The SetGenerate property specifies whether to generate mutators for relations. (Default = Checked)

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be

modified in any way.

- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for the element has been applied.

Default = "Default"

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

For example, you can use this property to add an #ifdef to indicate that an operation is available only when the code is compiled with _DEBUG, by setting the following properties:

- For SpecificationProlog, enter the value #ifdef _DEBUG and a new line.
- For SpecificationEpilog, enter the value #endif
- For ImplementationProlog, enter the value #ifdef _DEBUG and a new line.
- For ImplementationEpilog, enter the value #endif

Default = Blank

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to "abstract."

You must include the space after the word "abstract." If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...}

The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class Yes No Inside Package Yes Yes Inside

(empty MultiLine)

Static

The Static property is a Boolean value that determines whether class-wide relations are enabled. Class-wide members of a class are shared between all instances of that class and are mapped as static. When a relation is tagged as static:

- The data member is generated as static (with the static keyword).
- The relation accessors are generated as static.
- The mutators of directional relations are generated as static. The mutators of symmetric relations are generated as common (non-static) operations.

Note the following behavior and restrictions:

- If there are links between instances based on static relations, code generation will initialize all the valid links. In case of a limited relation size, the last initialization is preserved.
- When you generate instrumented code (animation or tracing), relation NOTIFY calls are not added to static relation mutators.
- Animation associates static relations with the class instances, not the class itself.
- In an instrumented application (animation or tracing), the static relations names appear in each instance node; however, the values of directional static relations are not visible.

You may also want to use the "Filter" facility in this window to refer to the definitions of these properties:

CG::Relation::Containment

Containertype::Relationtype::CreateStatic

Containertype::Relationtype::InitStatic

(Default = Cleared)

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. (Default = Public)

Solaris2

Environment settings (Compiler, framework libraries, etc.) for Solaris 2, using Sun compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/Solaris2)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches.

(Default = -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DSolaris2 $OMCPPCompileCommandSet -c)

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component. The default values are as follows:

(Default = -g)

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = -O)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default value is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

You may also want to use the "Filter" facility in this window to refer to the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is as follows:

ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

This default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is$(OMROOT)/LangC/lib/sol2WebComponents$(LIB_EXT), $(OMROOT)/lib/sol2WebServices$(LIB_EXT), -lsocket -lnsl.

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

The C default value is "include."

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = xterm -e $executable)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

(Default = $OMROOT/etc/sol2make $makefile $maketarget)

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = -g)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

(Default = -O)

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ################ ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute

from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################ ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables.

Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################# Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile. The default values are as follows:

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Checked)

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:])

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

The extension ".h" is the default for C.

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style.

If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

(Default = Checked)

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

(Default = Checked)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

Solaris2GNU

Environment settings (Compiler, framework libraries, etc.) for Solaris 2, using GCC compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/Solaris2)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CompileSwitches

The CompileSwitches property specifies the compiler switches.

(Default = -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT) -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf $(INST_FLAGS) $(INCLUDE_PATH) $(INST_INCLUDES) -DSolaris2 $OMCPPCompileCommandSet -c)

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath @$(CC) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component. The default values are as follows:

(Default = -g)

CPPCompileRelease

The CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = -O)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

The default value is as follows:

$OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

You may also want to use the "Filter" facility in this window to refer to the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property.

ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

The default value is as follows:

ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = Blank)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile.

This default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is$(OMROOT)/LangC/lib/sol2WebComponents$(LIB_EXT), $(OMROOT)/lib/sol2WebServices$(LIB_EXT), -lsocket -lnsl.

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is Checked.

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

The C default value is "include."

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

(Default = xterm -e $executable)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

(Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vxmake.bat\" $makefile $maketarget")

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = -g)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

(Default = empty string)

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode. The default values are as follows:

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE

CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################# ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################# Linking instructions ##################

```
############################################################
$(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath
$OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD)
$OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \
$(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT)
$(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo
Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS)
$(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase
$OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase
$(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase
$(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist
$(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)
```

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile. The default values are as follows:

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

The extension ".h" is the default for C.

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

Statechart

The Statechart metaclass contains the properties that control statechart code generation.

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify

can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be modified in any way.
- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for the element has been applied.

Default = "Default"

StatechartImplementation

Prior to version 7.3 of Rational Rhapsody, the transition-handling code generated by Rhapsody used a switch statement to represent the possible states. Beginning with version 7.3, this code uses an if/else structure. To allow older models to use the previous code generation behavior, a property called StatechartImplementation was added to the Pre73 backward compatibility profiles. The possible values for the property are:

- SwitchOnly - transition-handling code uses a switch statement to represent the possible states
- Default - the transition-handling code uses an if/else structure to represent the possible states

Default = SwitchOnly

StatechartStateOperations

This property determines whether the code is generated for this feature.

Rhapsody provides a mechanism for serialization of reactive instances. By setting a number of Rational Rhapsody properties, you can have methods added to the generated code, which you can then use to implement serialization.

The possible values for this property are as follows:

- None - code is not generated for the feature
- WithoutReactive - Rational Rhapsody does not generate calls to OMReactive
- WithReactive - Rational Rhapsody generates calls to OMReactive

(Default = None)

The other C properties used in the serialization methods are as follows:

- C_CG::Framework::ReactiveGetStateCall
- C_CG::Framework::ReactiveSetStateCall

- C_CG::Framework::ReactiveStateType

Type

The Type metaclass contains a property that affects the visibility of data types.

AnimEnumerationTypeImage

The AnimEnumerationTypeImage property is a Boolean value that determines whether the Image attribute is used for enumerated types when using animation. (Default = Cleared)

AnimSerializeOperation

The AnimSerializeOperation property enables you to specify the name of an external function used to animate all attributes and arguments that are of that type. Rhapsody can animate (display) the values of simple types and one-dimensional arrays without any problem. To display the current values of such attributes during an animation session, run the features window for the instance.

However, if you want to animate a more complex type, such as a date, the type must be converted to a string (char *) for Rhapsody to display it. This is generally done by writing a global function, an instrumentation function , that takes one argument of the type you want to display, and returns a char *. You must disable animation of the instrumentation function itself (using the Animate and AnimateArguments properties for the function).

For example, you can have a type tDate, defined as follows: typedef struct date { int day; int month; int year; } %s; You can have an object with an attribute count of type int, and an attribute date of type tDate. The object can have an initializer with the following body: me-date.month = 5; me-date.day = 12; me-date.year = 2000; If you want to animate the date attribute, the AnimSerializeOperation property for date must be set to the name of a function that will convert the type tDate to char *. For example, you can set the property to a function named showDate. This function name must be entered without any parentheses. It must take an attribute of type tDate and return a char *. The Animate and AnimateArguments properties for the showDate function must be set to False.

The implementation of the showDate function might be as follows: showDate(tDate aDate) { char* buff; buff = (char*) malloc(sizeof(char) * 20); sprintf(buff,"%d %d %d", aDate.month,aDate.day,aDate.year); return buff; }

When you run this model with animation, instances of this object will display a value of 5 12 2000 for the date attribute in the browser. If the showDate function is defined in the same class that the attribute belongs to and the function is not static, the AnimSerializeOperation property value should be similar to the following:

myReal-showDate

This value shows that the function is called from the serializeAttributes function, located in the class OMAnimatedclassname. The showDate function must allocate memory for the returned string via the malloc/alloc/calloc function in C, or the new operator in C++. Otherwise, the system will crash. (Default = empty string)

AnimUnserializeOperation

The AnimUnserializeOperation property converts a string to the value of an element (the opposite of the AnimSerializeOperation property). Unserialize functions are used for event generation or operation invocation using the Animation toolbar to convert the string (received from the user) to the value of the event or operation before the event generation or operation invocation.

For example, your serialization operation might look similar to the following:

char* myX2String(const Rec f) { char* cS = new char[OutputStringLength]; /* conversion from the Rec type to string */ return (cS); }

The unserialization operation would be:

Rec myString2X (char* C, Rec T) { T = new Trc; /* conversion of the string C to the Rec type */ delete C; return (T); }

(Default = empty string)

DeclarationPosition

The DeclarationPosition property specifies where the type declaration appears. The possible values are as follows:

- BeforeClassRecord - The type declaration appears before the class record (CR) declaration if CR has a visibility set to public, and before the class record forward declaration if CR has a visibility set to private.
- AfterClassRecord - The type declaration appears after the class record declaration if CR has a visibility set to public, and after the class record forward declaration if CR has a visibility set to private.
- StartOfDeclaration - The type declaration appears among the first declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the first declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.
- EndOfDeclaration - The type declaration appears among the last declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the last declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.

If the C_CG::Type::Visibility property is set to "Body", no matter the settings of C_CG::Type::DeclarationPosition property, the type declaration still appears in the package body. (Default = BeforeClassRecord)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name

- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description

Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified the element tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the C_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property C_CG::Configuration::DescriptionEndLine.

(Default = empty string)

EnumerationAsTypedef

The EnumerationAsTypedef property specifies whether the generated enum should be wrapped by a typedef. This property is applicable to enumeration types in C and C++.

(Default = Checked)

In

The In property specifies how code is generated when the type is used with an argument that has the modifier In. The C value "const $type*" is the default.

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier "InOut."

IsLimited

The IsLimited property determines whether the class or record type is generated as limited. (Default = Cleared)

LanguageMap

The LanguageMap property (specifies the Ada declaration for Rhapsody language-independent types. (Default = empty string)

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier Out. The default value is $type**.

PrivateName

The PrivateName property specifies the pattern used to generate names of private operations in C. (Default = $typeName)

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C. (Default = $objectName_$typeName)

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code. See the Rational Rhapsody Help for detailed information about using composite types. (Default = "")*

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type. (Default = $type)*

Simplify

If you are using the Rational Rhapsody customizable code generation mechanism, the property Simplify can be used to change the way specific types of elements are handled by Rhapsody when it transforms the model into a simplified model.

The property can take any of the following values:

- None - The element is ignored.
- Copy - The element will just be copied from the original to the simplified model. It will not be

modified in any way.

- Default - Uses the standard simplification for this item, as defined in Rational Rhapsody.
- ByUser - Uses the customized simplification provided by the user. (For details, see the section on User-Provided Simplification in the Rational Rhapsody Help.)
- ByUserPostDefault - Uses the customized simplification provided by the user, but only after Rhapsody's standard simplification for the element has been applied.

Default = "Default"

StructAsTypedef

The StructAsTypedef property specifies whether the generated struct should be wrapped by a typedef. This property is applicable to structure types in C and C++. (Default = Checked)

TriggerArgument

The TriggerArgument property is used for mapping event and triggered operation arguments to code instead of the In, InOut, and Out properties. A different property is required because of code generation limitations related to event arguments. You may also want to use the "Filter" facility in this window to refer to these definitions:

- In
- InOut
- Out

(Default = $type)

UnionAsTypedef

The UnionAsTypedef property specifies whether the generated union should be wrapped by a typedef. This property is applicable to union types in C and C++. (Default = Checked)

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

(Default = Public)

VxWorks

The VxWorks metaclass contains the Environment settings (Compiler, framework libraries, etc.) for VxWorks compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vxmake.bat" vxbuild.mak build 5.5 \"CPU=$BSP\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

-I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -O0 -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = vxmain)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vxWebComponents(CPU)(LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Checked)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = $OMROOT/DLLs/TornadoIDE.dll)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vxmake.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ################ ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True"

ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT)

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################ Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (make)(.)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example,

enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

VxWorks6diab

The VxWorks6diab metaclass contains the Environment settings (Compiler, framework libraries, etc.) for VxWorks6diab compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=diab\" \"TOOL_FAMILY=diab\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports

integration with Applied Microsystems Corporation CodeTest.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

$IgnoreSwitches -I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -O -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = vxmain)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable

created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vxWebComponents(CPU)(TOOL)$(LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(TOOL)$(LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Checked)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile

can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################## Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################ ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt

LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################# ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################ Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase

$OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of

compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and

RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

VxWorks6diab_RTP

The VxWorks6diab_RTP metaclass contains the Environment settings (Compiler, framework libraries, etc.) for VxWorks6diab compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link.

The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=diab\" \"TOOL_FAMILY=diab\" \"BUILD=$BuildCommandSet\" \"DISTRIBUTED=TRUE\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

-I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .vxe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vx$(DIST_PREFIX)WebComponents(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Cleared)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = empty string)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP diab"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ################ ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True"

ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT)

$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################ Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = (["]([^:]+)["][,][ ]line ([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (["]([^:]+)["][,][ ]line ([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (["]([^:]+)["][,][ ]line ([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example,

enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

VxWorks6gnu

The VxWorks6gnu metaclass contains the Environment settings (Compiler, framework libraries, etc.) for VxWorks6gnu compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=gnu\" \"TOOL_FAMILY=gnu\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports

integration with Applied Microsystems Corporation CodeTest.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

-I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -O0 -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = vxmain)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable

created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vxWebComponents(CPU)(LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Checked)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = -g)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################## Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################## ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT

C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################# ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################ Linking instructions ################# ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS)

$(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error|warning)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of

compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example, enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and

RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

VxWorks6gnu_RTP

The VxWorks6gnu_RTP metaclass contains the Environment settings (Compiler, framework libraries, etc.) for the VxWorks6gnu_RTP compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=gnu\" \"TOOL_FAMILY=gnu\" \"BUILD=$BuildCommandSet\" \"DISTRIBUTED=TRUE\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Cleared

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest™.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

-I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -O0 -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .vxe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vx$(DIST_PREFIX)WebComponents(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Cleared)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = empty string)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = -g)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet -MD -MP)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .mak)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ################ ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True"

ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables.

Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE

INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################## Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error|warning)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

ParseMakeError

The property ParseMakeError is used to define a regular expression that represents the format of make process or linker error messages. This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(make)(.*)(Error)*

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (error)

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (warning)

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

PathWhiteSpaceHandling

For different operating systems, there are different methods for handling spaces in file paths, for example,

enclosing the entire path in quotation marks. The property PathWhiteSpaceHandling allows you to specify the method that should be used for a given environment. The possible values are:

- NoHandling - the path should be left as is, with no special handling for spaces
- SurroundingQuotes - the entire path should be enclosed in quotation marks
- BackslashBeforeSpace - spaces in paths should be preceded by backslashes, as is the practice in VxWorks platforms

Default = BackslashBeforeSpace

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

(Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as

Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

WorkbenchManaged

The WorkbenchManaged metaclass contains the Environment settings (Compiler, framework libraries, etc.) for WorkbenchManaged compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files,

and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

AutoAttachToIDEDebugger

The property AutoAttachToIDEDebugger is used to specify that you would like the Workbench debugger to be automatically synchronized with the Rational Rhapsody animation. If for some reason you do not want automatic synchronization, for example, if you prefer to manually connect to the relevant target server, then you can set the value of this property to False.

Default = Checked

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file.

Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=$Tool\" \"TOOL_FAMILY=$Tool\" \"BUILD=$BuildCommandSet\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Checked

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

$IgnoreSwitches -I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -O -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = vxmain)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .out)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vxWebComponents(CPU)(TOOL)$(LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(TOOL)$(LIB_EXT)).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Checked)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = -g)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet -MD -MP)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .makefile)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ############### ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section

of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################## ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables.

Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS=

INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################## Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ([^:]+)[:]([0-9]+)[:])

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application.

To run remotely, the UseRemoteHost property must be set to True. If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

(Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

(Default = Checked)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

WorkbenchManaged_RTP

The WorkbenchManaged_RTP metaclass contains the Environment settings (Compiler, framework libraries, etc.) for WorkbenchManaged_RTP compiler.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path.

Previously, the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files.

To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

(Default = $(OMROOT)/LangC/osconfig/VxWorks)

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody does not allow you to use.

In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting.

Note that this property affects the algorithm only when the active configuration is of the selected environment.

(Default = empty string)

AutoAttachToIDEDebugger

The property AutoAttachToIDEDebugger is used to specify that you would like the Workbench debugger to be automatically synchronized with the Rational Rhapsody animation. If for some reason you do not want automatic synchronization, for example, if you prefer to manually connect to the relevant target server, then you can set the value of this property to False.

Default = Checked

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property.

(Default = PENTIUM)

BuildArgumentsInIDE

The property BuildArgumentsInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. The property allows you to enter a string of arguments that is used as build arguments if you are not using the IDE default build command.

The arguments provided here are only used if the value of the property

C_CG:[environment]:BuildCommandInIDE is not an empty string.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandInIDE

The property BuildCommandInIDE is used when building an application in an IDE that has been integrated with Rational Rhapsody, such as Eclipse. If this property is left blank, the IDE default build command is used. If you enter a different string, then the command you entered is used when building the application rather than the IDE default build command.

Keep in mind that the build is performed by the IDE only if the value of the property BuildInIDE is set to True.

Default = Blank

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

(Default = Debug)

buildFrameworkCommand

The Code menu in Rational Rhapsody includes an option called Build Framework. When you select this option, Rational Rhapsody rebuilds its framework libraries for the environment specified on the Settings tab of the Features dialog for the active configuration. The property buildFrameworkCommand is used to specify the command that should be carried out when the Build Framework option is selected.

Default = "$OMROOT/etc/Executer.exe" "\""$OMROOT\etc\vx6make.bat" vxbuild.mak buildLibs 6.5 \"CPU=$BSP\" \"TOOL=$Tool\" \"TOOL_FAMILY=$Tool\" \"BUILD=$BuildCommandSet\" \"DISTRIBUTED=TRUE\" \" "

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = Checked

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation CodeTest.

(Default = CC = $(AMC_HOME)\bin\ctcc)

CompileSwitches

The CompileSwitches property specifies the compiler switches.

The default value is as follows:

$IgnoreSwitches -I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangC -I$(OMROOT)/LangC/oxf -DVxWorks $(INST_FLAGS) $(INCLUDE_PATH) $OMCPPCompileCommandSet -Wno-unused -c

CPPCompileCommand

The CPPCompileCommand property is a string that enables you to specify a different compile command. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

The default value is as follows:

@echo Compiling $OMFileImpPath $(CREATE_OBJ_DIR) @$(CC) $(CFLAGS) $OMFileCPPCompileSwitches -o $OMFileObjPath $OMFileImpPath

CPPCompileDebug

The CPPCompileDebug property modifies the makefile compile command with switches for building a debug version of the component.

(Default = -O -g)

CPPCompileRelease

TThe CPPCompileRelease property enables you to specify additional compilation flags for a configuration

set to Release mode.

(Default = empty string)

DependencyRule

The DependencyRule property specifies how file dependencies for a configuration are generated in the makefile.

For example, the following dependency rule lists the file dependencies for a Windows application with a GUI, including bitmaps, icons, and resource files: $OMFileObjPath : $OMFileImpPath "*.bmp" "*.ico" "*.rc2"

The default value is $OMFileObjPath : $OMFileImpPath $OMFileSpecPath $OMFileDependencies.

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command.

(Default = Checked)

EnableDebugIntegrationWithIDE

When using Rhapsody in conjunction with an IDE such as Eclipse, the property EnableDebugIntegrationWithIDE can be used to specify whether or not the IDE debugger should be used in conjunction with the Rational Rhapsody animation feature.

If the value of the property is set to True, the IDE debugger is used.

Default = Cleared

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

(Default = main)

Use the "Filter" facility in this window to see the definition of the EntryPointDeclarationModifier property for more information.

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line

number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

(Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2)

ExeExtension

The ExeExtension property is used to specify the file extension you would like to use for the executable created by Rhapsody.

Note that the full name of the executable is composed of the value of the property C_CG::<Environment>::ExeName plus the value of this property.

(Default = .vxe)

ExeName

By default, the name of the executable created by Rhapsody is the name of the active component. If you would like to use a different name for the executable, enter the name as the value of the property ExeName.

If you leave the value of the property blank, the name of the active component is used.

The name provided for this property is used both for executables and for libraries.

Note that the full name of the executable is composed of the value of this property plus the value of the property C_CG::<Environment>::ExeExtension.

(Default = Blank)

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements.

The file inclusions are generated in the makefile.

The default value is $OMSpecIncludeInElements $OMImpIncludeInElements.

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with

Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile.

Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

This default value is $(OMROOT)/LangC/lib/vx$(DIST_PREFIX)WebComponents(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT), $(OMROOT)/lib/vxWebServices(CPU)(RTP_SUFFIX)$(TOOL)$(RHP_LIB_EXT).

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled.

If IDE support is enabled (Checked), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to Cleared, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = Checked)

IDEInterfaceDLL

The IDEInterfaceDLL property is a string that points to the IDE adapter DLL. You should not have any reason to modify this property. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody.

(Default = $OMROOT/DLLs/WorkbenchDebuggerIDE.dll)

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values are as follows:

(Default = .c)

Include

The Include property specifies the environment-specific command that is generated in the makefile to include other makefiles.

(Default = include)

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

The default value is as follows:

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget"

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

(Default = Cleared)

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

(Default = .a)

LinkDebug

The LinkDebug property specifies the special link switches used to link in debug mode.

(Default = empty string)

LinkRelease

The LinkRelease property specifies the special link switches used to link in release mode.

LinkSwitches

The LinkSwitches property specifies the standard link switches used to link in any mode.

(Default = $OMLinkCommandSet)

MakeExtension

The property MakeExtension can be used to specify the file extension you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the extension that you would like to use. Note that the first part of the filename can be customized by modifying the value of the property C_CG::<Environment>::MakeFileName.

If you do not want Rhapsody to add a file extension, leave the value of this property blank.

(Default = .makefile)

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration. For example, the default content of the target type section of a C++ makefile for the Microsoft environment is as follows: ############# Target type (Debug/Release) ################ ## CPPCompileDebug=$OMCPPCompileDebug CPPCompileRelease=$OMCPPCompileRelease LinkDebug=$OMLinkDebug LinkRelease=$OMLinkRelease BuildSet=$OMBuildSet SUBSYSTEM=$OMSubSystem COM=$OMCOM RPFrameWorkDll=$OMRPFrameWorkDll ConfigurationCPPCompileSwitches= $OMReusableStatechartSwitches $OMConfigurationCPPCompileSwitches !IF "$(RPFrameWorkDll)" == "True" ConfigurationCPPCompileSwitches= $(ConfigurationCPPCompileSwitches) /D "FRAMEWORK_DLL" !ENDIF !IF "$(COM)" == "True" SUBSYSTEM=/SUBSYSTEM:windows !ENDIF

Compilation Flags The compilation flags section of the makefile contains the default compilation flags stored in the CompileSwitches property. For example, the default content of the compilation flags section of a C++ makefile for the Microsoft environment is as follows: ################### Compilation flags ################### ## INCLUDE_QUALIFIER=/I LIB_PREFIX=MS

Commands Definitions The commands definition section of the makefile specifies programs to execute

from the makefile. For example, the default commands definition section of a C++ makefile for the Microsoft environment is as follows: ############ Commands definition ################ ## RMDIR = rmdir LIB_CMD=link.exe -lib LINK_CMD=link.exe LIB_FLAGS=$OMConfigurationLinkSwitches LINK_FLAGS=$OMConfigurationLinkSwitches $(SUBSYSTEM) / MACHINE:I386

Generated Macros The generated macros section of the makefile contains a variable that expands to the Rational Rhapsody -generated macros in the makefile. For example: ############### Generated macros ################# ## $OMContextMacros OBJ_DIR=$OMObjectsDir !IF "$(OBJ_DIR)"!="" CREATE_OBJ_DIR=if not exist $(OBJ_DIR) mkdir $(OBJ_DIR) CLEAN_OBJ_DIR= if exist $(OBJ_DIR) $(RMDIR) $(OBJ_DIR) !ELSE CREATE_OBJ_DIR= CLEAN_OBJ_DIR= !ENDIF

The $OMContextMacros keyword expands several macros in the makefile. Each makefile macro has its own keyword. You can use these keywords separately to customize the makefile. The $OMContextMacros variable enables you to modify target-specific variables. Replace the $OMContextMacros line in the MakeFileContent property with the following: FLAGSFILE=$OMFlagsFile RULESFILE=$OMRulesFile OMROOT=$OMROOT C_EXT=$OMImplExt H_EXT=$OMSpecExt OBJ_EXT=$OMObjExt EXE_EXT=$OMExeExt LIB_EXT=$OMLibExt INSTRUMENTATION=$OMInstrumentation TIME_MODEL=$OMTimeModel TARGET_TYPE=$OMTargetType TARGET_NAME=$OMTargetName $OMAllDependencyRule TARGET_MAIN=$OMTargetMain LIBS=$OMLibs INCLUDE_PATH=$OMIncludePath ADDITIONAL_OBJS=$OMAdditionalObjs OBJS= $OMObjs

Predefined Macros The predefined macros section of the makefile contains other macros than the Rational Rhapsody -generated macros specified in the generated macros section. For example, the default predefined macros section of a C++ makefile for the Microsoft environment is as follows: ################### Predefined macros ################ ## $(OBJS) : $(INST_LIBS) $(OXF_LIBS) LIB_POSTFIX= !IF "$(BuildSet)"=="Release" LIB_POSTFIX=R !ENDIF !IF "$(TARGET_TYPE)" == "Executable" LinkDebug=$(LinkDebug) /DEBUG LinkRelease=$(LinkRelease) /OPT:NOREF !ELSEIF "$(TARGET_TYPE)" == "Library" LinkDebug=$(LinkDebug) /DEBUGTYPE:CV !ENDIF !IF "$(INSTRUMENTATION)" == "Animation" INST_FLAGS=/D "OMANIMATOR" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfanimdll$(LIB_POSTFIX) $(LIB_EXT) !ELSE INST_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomanim$(LIB_POSTFIX) (LIB_EXT) OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB POSTFIX)$(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "Tracing" INST_FLAGS=/D "OMTRACER" INST_INCLUDES=/I $(OMROOT)\LangCpp\aom /I $(OMROOT)\LangCpp\tom !IF "$(RPFrameWorkDll)" == "True" INST_LIBS= OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxftracedll$(LIB_POST FIX)$(LIB_EXT) !ELSE INST_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)tomtrace$ (LIB_POSTFIX) $(LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)aomtrace$(LIB_POSTFIX) $(LIB_EXT) OXF_LIBS= $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfinst$(LIB_POSTFIX) (LIB_EXT) $(OMROOT)\LangCpp\lib\$(LIB_PREFIX)omComAppl$(LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB=wsock32.lib !ELSEIF "$(INSTRUMENTATION)" == "None" INST_FLAGS= INST_INCLUDES= INST_LIBS= !IF "$(RPFrameWorkDll)" == "True" OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxfdll$ (LIB_POSTFIX)$(LIB_EXT) !ELSE OXF_LIBS=$(OMROOT)\LangCpp\lib\$(LIB_PREFIX)oxf$ (LIB_POSTFIX)$(LIB_EXT) !ENDIF SOCK_LIB= !ELSE !ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified. !ENDIF

Generated Dependencies The generated dependencies section of the makefile contains a variable that expands to Rational Rhapsody -generated dependencies and compilation instructions. For example, the generated dependencies section of a C++ makefile for the Microsoft environment is as follows: ########## Generated dependencies ################## ## $OMContextDependencies $OMFileObjPath : $OMMainImplementationFile $(OBJS) $(CPP) $(ConfigurationCPPCompileSwitches) /Fo"$OMFileObjPath" $OMMainImplementationFile

Linking Instructions The linking instructions section of the makefile contains the predefined linking instructions. For example, the default linking instructions section of a C++ makefile for the Microsoft environment is as follows: ################# Linking instructions ################## ## $(TARGET_NAME)$(EXE_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMFileObjPath $OMMakefileName $OMModelLibs @echo Linking $(TARGET_NAME)$(EXE_EXT) $(LINK_CMD) $OMFileObjPath $(OBJS) $(ADDITIONAL_OBJS) \ $(LIBS) \ $(INST_LIBS) \ $(OXF_LIBS) \ $(SOCK_LIB) \ $(LINK_FLAGS) /out:$(TARGET_NAME)$(EXE_EXT) $(TARGET_NAME)$(LIB_EXT) : $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName @echo Building library $@ $(LIB_CMD) $(LIB_FLAGS) /out:$(TARGET_NAME)$(LIB_EXT) $(OBJS) $(ADDITIONAL_OBJS) clean: @echo Cleanup $OMCleanOBJS if exist $OMFileObjPath erase $OMFileObjPath if exist *$(OBJ_EXT) erase *$(OBJ_EXT) if exist $(TARGET_NAME).pdb erase $(TARGET_NAME).pdb if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT) if exist $(TARGET_NAME).ilk erase $(TARGET_NAME).ilk if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT) $(CLEAN_OBJ_DIR)

MakeFileName

The property MakeFileName can be used to specify the filename you would like to use for the makefile generated by Rhapsody. For the value of this property, enter the name that you would like to use for the file.

Note that this property only specifies the first part of the filename. The extension is specified using the property C_CG::<Environment>::MakeExtension.

If the property value is left blank, Rational Rhapsody uses the name of the component.

(Default = Blank)

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

(Default = NULL)

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

(Default = $(RM) $OMFileObjPath)

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile.

(Default = empty string)

ObjectsDirectory

The ObjectsDirectory property specifies an alternate name for the directory for compiled object files in the generated makefile.

(Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

(Default = .o)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

(Default = Cleared)

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning)

Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

(Default = ["]([^:]+)["][,][ ]line ([0-9]+)[:])

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

(Default = /)

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

(Default = Checked)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True.

If UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine. (Default = empty string)

ReusableStatechartSwitches

The ReusableStatechartSwitches property defines the compilation switch that was added to the makefile to support reusable statecharts. See the upgrade history on the support site for detailed information on this change.

(Default = -DOM_REUSABLE_STATECHART_IMPLEMENTATION)

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

(Default = .h)

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIXstyle path names.

The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

(Default = Checked)

UpdateBuildSettingsInIDE

The property UpdateBuildSettingsInIDE is used when using Rhapsody in conjunction with an IDE such as Eclipse. If the value of the property is set to True, then Rhapsody updates the build settings in the IDE after any changes are made to the build settings (such as make-related properties). The update is performed after code generation.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle.

(Default = Checked)

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

(Default = Checked)

C_ReverseEngineering

In addition to the ReverseEngineering subject, Rational Rhapsody provides language-specific subjects to control how Rhapsody imports legacy code. Most of the properties are identical for each language. Any language-specific properties are clearly labeled. In general, most of the reverse engineering (RE) properties have graphical representation in the Reverse Engineering Options window. You should change the options using this window instead of the corresponding properties. The metaclasses are as follows:

- Filtering
- ImplementationTrait
- Main
- Parser
- Promotions

Filtering

The Filtering metaclass contains properties that control which items are analyzed during the reverse engineering operation.

AnalyzeGlobalFunctions

The AnalyzeGlobalFunctions property specifies whether to analyze global functions.

Default = Checked

AnalyzeGlobalTypes

The AnalyzeGlobalTypes property specifies whether to analyze global types.

Default = Checked

AnalyzeGlobalVariables

The AnalyzeGlobalVariables property specifies whether to analyze global variables.

Default = Checked

CreateReferenceClasses

The CreateReferenceClasses property specifies whether to create external classes for undefined classes that result from forward declarations and inheritance. By default, reference classes are created (as in previous versions of Rational Rhapsody). If the incomplete class cannot be resolved, the tool deletes the

incomplete class if this property is set to Cleared. In some cases, the class cannot be deleted (for example, a class referenced by a typedef type).

Default = Checked

IncludeInheritanceInReference

The IncludeInheritanceInReference property specifies whether to include inheritance information in reference classes.

Default = Cleared

ReferenceClasses

The ReferenceClasses property specifies which classes to model as reference classes. Reference classes are classes that can be mentioned in the final design as placeholders without having to specify their internal details. For example, you can include the MFC classes as reference classes, without having to specify any of their members or relations. They would simply be modeled as terminals for context, to show that they are acting as superclasses or peers to other classes.

Default = empty string

ReferenceDirectories

The ReferenceDirectories property specifies which directories (and subdirectories) contain reference classes.

Default = empty string

ImplementationTrait

The ImplementationTrait metaclass contains properties that determine the implementation traits used during the reverse engineering operation.

AnalyzeIncludeFiles

The AnalyzeIncludeFiles property specifies which, if any, include files should be analyzed during reverse engineering. The possible values are as follows:

- AllIncludes - Analyze all include files.
- IgnoreIncludes - Ignore all include files.
- OnlyFromSelected - Analyze the specified include files only.
- OnlyLogicalHeader - Analyze the logical header files only.

(C Default = AllIncludes)

AutomaticIncludePath

When Rhapsody reverse engineers a file, there may be cases where the file references a header file but the path in the include directive is not clear enough for Rhapsody to find the file. If you set the value of the property AutomaticIncludePath to Checked, then in such cases, Rational Rhapsody will search the list of files to be reverse engineered to see if the list contains a header file with that name. If there is such a file, Rational Rhapsody uses the full path that was provided for that header file, assuming that this is the header file that was being referenced in the original file.

Rhapsody performs this search for ambiguous header files when it does macro collection. This means that if the value of the property C_ReverseEngineering::ImplementationTrait::CollectMode is set to None, then Rational Rhapsody will not search for ambiguous header files even if the value of the property AutomaticIncludePath is set to Checked.

Default = Checked

CreateBlackDiamondAssociations

The property CreateBlackDiamondAssociations specifies how the reverse engineering feature should handle composition relationships. If the value of the property is set to False, then Rational Rhapsody creates parts. If the value of the property is set to Checked, Rational Rhapsody creates composition associations (black diamond).

Default = Cleared

CreateDependencies

The CreateDependencies property is used during reverse engineering (RE) for creating dependencies from include statements found in the imported code. This property determines whether the RE utility creates dependencies. Reverse engineering imports include statements as dependencies if the option Create Dependencies from Includes is set in the Rational Rhapsody GUI. This operation is successful if the reverse engineering utility analyzes both the included file and the source - and the source and included files contain class declarations for creating the dependencies between them. If there is not enough information, the includes are not converted dependencies. This can happen in the following cases:

- The include file was not found, or is not in the scope Input tab settings.
- A class is not defined in the include file or source file, so the dependency could not be created.

If the dependency is not created successfully, the include files that were not converted to dependencies are imported to the C_CG::Class::SpecIncludes or ImpIncludes properties so you do not have to re-create them manually. If the include file is in the specification file, the information is imported to the SpecIncludes property; if it is in the implementation file, the information is imported to the ImpIncludes property. If a file contains several classes, include information is imported for all the classes in the file. The possible values for this property are as follows:

- None - Nothing is imported from include statements.
- DependenciesOnly - Model dependencies are created from include statements when it is possible to do

so. This is the RE behavior of previous versions of Rational Rhapsody.

- All - The reverse engineering utility attempts to map the include file as a dependency. If it fails, the information is written to a property.

In previous versions of Rational Rhapsody, this property was a Boolean value. For backward compatibility, the old values are mapped as follows:

Old Value New Value Checked DependenciesOnly Cleared None

1. In addition to influencing reverse engineering, the CreateDependencies property also impacts the reverse engineering of user code added to model elements. The rules for interpreting #include and friend declarations for reverse engineering are as follows:

- Any #include OTHER in FILE is represented as a Uses dependency between each (outer) packages or classes in FILE to any (outer) packages or class in OTHER.
- If OTHER is not a specification file, the information is lost.
- If FILE is a specification file, the RefereeEffect is Specification. If FILE is an implementation file, the RefereeEffect is Implementation. Otherwise, the information is lost.

2. Any forward of a class or a package (via namespace) E in FILE is represented as a Uses dependency between each (outer) packages/classes in FILE to E. The RefereeEffect is Existence.

3. This dependency is not added, if a Uses dependency can be matched.

4. Redundant Uses dependencies are removed. For example, when a relation is synthesized from a pointer to B, it is not necessary to add a Uses dependency.

5. A friend F (only when F is a class) of class C is represented as a dependency with DependencyType to be Friendship from F to C.

Default = All)

CreateFilesIn

The CreateFilesIn property is a placeholder for the reverse engineering option Create File-s In option. See the Rational Rhapsody Help for more information. You should not set this value directly. The default value for C is Package.

CollectMode

The CollectMode property allows Rhapsody to collect macros. The possible values are as follows:

- None - Macros are not collected from include files that are not on the reverse engineering list.
- Once - Macros are collected only if the model does not yet include a controlled file of collected macros.
- Always - Macros are collected each time reverse engineering is carried out. The controlled file that stores the macros are replaced each time.

(C Default = None)

DataTypesLibrary

The Mapping tab of the Reverse Engineering Options dialog allows you to specify a list of types that should be modeled as "Language" types. You can add individual types to the list or groups of types that you have previously defined as data types for a specific library.

If you select the option of adding a library, you are presented with a drop-down list of libraries to choose from. The libraries on this list are taken from the value of the property DataTypesLibrary. You can add a number of libraries to the drop-down list by using a comma-separated list of names as the value for this property.

When you select a library from the drop-down list, all of the types that were defined for that library are added to the list of types.

You define types for a library by carrying out the following steps:

- In the relevant .prp file, under the subject [lang]_ReverseEngineering, add a metaclass with the name of the library (using the same name you used in the value of the property DataTypesLibrary).
- Under the new metaclass, add a property called DataTypes.
- For the value of the DataTypes property that you added, enter a comma-separated list of the types that you want to include for that library.
- Now, if you select the library from the drop-down list displayed on the Mapping tab, the types you defined with the DataTypes property is automatically added to the list of types that should be modeled as "Language" types.

Default = Blank

ImportAsExternal

The property ImportAsExternal specifies whether the elements contained in the files you are reverse engineering should be brought into the model as "external" elements. This means that code will not be generated for these elements during code generation.

This property corresponds to the Import as External check box on the Mapping tab of the Reverse Engineering Options dialog.

Default = Cleared

ImportDefineAsType

The ImportDefineAsType property is a Boolean value that specifies how to import a #define. Note that models created before Version 5.2 automatically have this property overridden (set to True) when the model is loaded. The possible values are as follows:

- True - Import a #define as a user type.
- False - Import a #define as a constant variable, constant function, or type according to the following policy:
- If the #define has parameters, Rational Rhapsody creates a constant function. This applies to Rational

Rhapsody Developer for C only.

- If the #define does not have parameters and its value includes only one line, Rational Rhapsody creates a constant variable. In Rational Rhapsody Developer for C++, the property CG::Attribute::ConstantVariableAsDefine is set to True.
- If the #define was not imported as a variable or function, Rational Rhapsody creates a type (the behavior of Rational Rhapsody 5.0.1).

Default = False

ImportGlobalAsPrivate

The ImportGlobalAsPrivate property allows you to import C functions as public or private. The possible values are as follows:

- Never - Import globals (functions) as public. The declaration remains in the specification file.
- InImplementation - Global functions are imported as private. Both the declaration and the implementation of the function are imported into the implementation (.c) file.
- StaticInImplementation - Globals are imported as private in the implementation (.c) file and the functions are marked as static. (same as "InImplementation" but the keyword "static" is added to the declaration and implementation of the function).

ImportStructAsClass

The ImportStructAsClass property is a Boolean value specifies how structs in external code are imported during reverse engineering. The possible values are as follows:

- Checked - structs are imported as classes (as in Rational Rhapsody 5.0 and earlier).
- Cleared - structs are imported as types of kind Structure.

Default = Cleared

LocalizeRespectInformation

When reverse engineering code in Respect mode, Rational Rhapsody stores information such as the order of code elements so that when code is regenerated from the model, the code will resemble as much as possible the original code.

When the property LocalizeRespectInformation is set to Checked, Rational Rhapsody stores this information as SourceArtifact elements below the relevant class. (These elements are not visible by default, but you can see them in the model if you set the value of the property ShowSourceArtifacts to True.)

If the value of the property LocalizeRespectInformation is set to Cleared, then Rational Rhapsody will store this "respect" information as File elements under the relevant Component.

Default = Checked

MacroExpansion

Early versions of Rational Rhapsody were not capable of importing macros in code such that they would be regenerated as macros. Rather, the code represented by the macro was stored in the model, and when the code was regenerated, the macro calls would be replaced with the relevant code.

Now, by default, Rational Rhapsody imports macros such that when the code is regenerated, the macro definition and macro calls are generated as they appeared in the original code that was reverse engineered.

If you would like the previous Rhapsody behavior, that is, replacement of macro calls with the actual macro code, you can set the MacroExpansion property to Checked.

Note that the property C_ReverseEngineering::Parser::ForceExpansionMacros allows you to specify that individual macros should be expanded during reverse engineering even if the value of the property MacroExpansion is set to False.

Default = Cleared

MapGlobalsToComponentFiles

The property MapGlobalsToComponentFiles allows you to specify whether Rational Rhapsody should map global variables, functions, and types to component files, reflecting the original file location of these elements in the files that were reverse engineered. The property can take any of the following values:

- OnExternal - Global variables, functions, and types should be mapped to component files only if the user selected the reverse engineering option "Import as External"
- TypesOnly - Global types should be mapped to component files, but not global variables and functions
- TypesOnExternal - Only global types should be mapped to component files, and this should only be done if the user selected the reverse engineering option "Import as External"
- False - Global variables, functions, and types should not be mapped to component files

Default = TypesOnExternal

MapToPackage

The property MapToPackage allows you to specify how the code elements you are reverse engineering should be divided into packages.

The property represents the options that appear in the Map to Package section of the Mapping tab in the Reverse Engineering Options dialog.

When the value of the property is set to Directory, a separate package is created for each subdirectory in the directory you have chosen to reverse engineer. The elements found in the files in each subdirectory is added to the package that corresponds to that subdirectory.

If you set the value of this property to User, then Rational Rhapsody will put all reverse engineered elements into a single package in the model. The name of the package is taken from the property [lang]_ReverseEngineering::ImplementationTrait::UserPackage.

Default = Directory

ModelStyle

The property ModelStyle determines how model elements are opened in the browser after reverse engineering - using a file-based functional approach or using an object-oriented approach based on classes (the corresponding property values are Functional and ObjectBased).

This property corresponds to the Modeling Policy radio buttons on the Mapping tab of the Reverse Engineering Options window.

Note that for C++ and Java, the file-based approach can only be used for visualization purposes. Rhapsody will not generate code from the model for elements imported using the Functional option. (You will notice that in the Reverse Engineering Options window, you can only select the File radio button if you first select the Visualization Only option.)

Default = Functional in RiC, ObjectBased in RiC++ and RiJ

PackageForExternals

If the value of the property UsePackageForExternals is set to Checked, the Rational Rhapsody reverse engineering feature puts all external elements in a separate package. You can control the name of this package by changing the value of the property PackageForExternals.

Default = Externals

PreCommentSensibility

During reverse engineering, a comment that comes immediately before the code for an element is considered a comment for that element, and the comment text is brought into Rational Rhapsody as the description for that element.

The property PreCommentSensibility is used to specify the maximum number of lines by which a comment can precede the code for an element and still be considered a comment for that element. Any comment that precedes an element by more than the number of lines specified is considered a global comment.

A value of 1 means that a comment must appear on the line prior to the code for an element to be considered a comment for that element.

Default = 2

ReflectDataMembers

The property ReflectDataMembers determines how the visibility of attributes is brought into the model when code is reverse engineered. The property affects both the visibility of the attribute in the regenerated code and the generation of get and set operations for the attribute. The property can take any of the following values:

- None - The visibility used for attributes is the same as that specified in the code that was reverse engineered. However, Rational Rhapsody generates public get/set operations for the attributes regardless of the visibility specified.
- VisibilityOnly - The visibility used for attributes is the same as that specified in the code that was reverse engineered. In addition, Rational Rhapsody generates get/set operations for the attribute with the same visibility. For example, if an attribute's visibility in the original code was private, the visibility is private in the regenerated code and the code will also include private get/set operations for the attribute.
- VisibilityAndHelpers - The visibility used for attributes is the same as that specified in the code that was reverse engineered. Rhapsody will not generate get/set operations for the attribute if the original code did not contain such operations.

Note that when the property is set to VisibilityAndHelpers, not only will get/set operations not be generated for attributes, but Rational Rhapsody does not generate any of its automatically-generated operations such as default constructors.

Default = VisibilityAndHelpers

RespectCodeLayout

The property RespectCodeLayout determines to what degree Rational Rhapsody attempts to save information about the code that is reverse engineered so that it is possible to match the original code when code is later regenerated from the model. This includes things like:

- order of #includes and other code elements
- handling of preprocessor directives such as #ifdefs
- keeping macro calls as they were rather than expanding the macro in the regenerated code
- handling of global comments

The property can take any of the following values:

- None - Rational Rhapsody does not save information about the order of elements in the code that is imported, nor does it save the information necessary to regenerate all elements back to the files from which they were originally imported.
- Mapping - Rational Rhapsody saves partial information so that it can regenerate all elements back to the files from which they were originally imported.
- Ordering - Rational Rhapsody saves all the information it can so that the regenerated code will match the original code as much as possible. See the examples listed above.

Note that even if the value of this property is set to Ordering, Rational Rhapsody will only attempt to match the regenerated code to the original code if the property [lang]_CG::Configuration::CodeGeneratorTool is set to Advanced, which is the default value for that property.

Default = Ordering

RootDirectory

This property specifies the root directory for reverse engineering. This root directory may contain all the

folders that should become package during the reverse engineering process. Rhapsody builds the package hierarchy according to the folder tree from the specified path.

Default = empty string

UseCalculatedRootDirectory

This property controls the use of the <lang>_ReverseEngineering::Implementation::RootDirectory property.

The possible values are:

- Never - Do not calculate the root directory.
- Always - Calculate the root directory and override the RootDirectory property.
- Auto - Ask the user if they want to override the value in the RootDirectory property if it is different from the calculated root directory. If the RootDirectory property is empty, Rational Rhapsody uses the calculated value without asking. This is the default value.

Default = Auto

UsePackageForExternals

When Rhapsody generates code, it does not regenerate code for elements that have been brought in as "external" elements. By default, the reverse engineering feature puts all external elements in a separate package in the model. You can change this behavior by changing the value of the property UsePackageForExternals. When a separate package is used, the name of the package is taken from the value of the property PackageForExternals.

Default = Checked

UserDataTypes

The UserDataTypes specifies classes to be modeled as data types. This property corresponds to types entered in the Add Type window.

Default = empty string

UserPackage

When reverse engineering files, Rational Rhapsody allows you the option of having packages created for each subdirectory or having all of the reverse-engineered elements placed in a single package. This option is controlled by the property [lang]_ReverseEngineering::ImplementationTrait::MapToPackage.

When MapToPackage is set to "User", you can use the property UserPackage to provide the name that you would like Rhapsody to use for the single package that will contain all of the reverse-engineered elements.

You can specify a nested package by using the following syntax: package1::package2

If the model already contains a package with the specified name, the reverse-engineered elements are put in that package. If not, Rational Rhapsody will create the package.

This property corresponds to the text field provided for the package name in the Map to Package section of the Mapping tab in the Reverse Engineering Options dialog.

Default = ReverseEngineering

Main

The metaclass Main contains properties that define the file extensions used for filtering files in the reverse engineering file selection dialog, as well as properties that enable jumping to problematic lines of code by double-clicking messages in the Output window.

ErrorMessageTokensFormat

When errors are encountered during reverse engineering, they are displayed in the Rational Rhapsody Output window. If you double-click the error message, you are taken to the problematic line in the relevant source file.

This ability is made possible by the values provided for the properties ParseErrorMessage and ErrorMessageTokensFormat.

The value of the property ParseErrorMessage is a regular expression that extracts the relevant filename and line number information from the Rational Rhapsody -generated error message. The value of the property ErrorMessageTokensFormat is then used to interpret the information that was extracted from the error message.

The value of the property ErrorMessageTokensFormat consists of a comma-separated list of keyword-value pairs representing the number of tokens contained in the extracted information, which token represents the filename, and which token represents the line number.

Users should not change the value of this property.

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ImplementationExtension

The ImplementationExtension property specifies the file extensions used to filter the list of files displayed in the Add Files window of the reverse engineering tool.

(C Default = .c)

ParseErrorMessage

When errors are encountered during reverse engineering, they are displayed in the Rational Rhapsody Output window. If you double-click the error message, you are taken to the problematic line in the relevant source file.

This ability is made possible by the values provided for the properties ParseErrorMessage and ErrorMessageTokensFormat.

The value of the property ParseErrorMessage is a regular expression that extracts the relevant filename and line number information from the Rational Rhapsody -generated error message. The value of the property ErrorMessageTokensFormat is then used to interpret the information that was extracted from the error message.

Users should not change the value of this property.

Default = "([a-zA-Z_]+[:0-9a-zA-Z_.\/])"[:][ ]*LINE[ ]*([0-9]+)*

SpecificationExtension

The property SpecificationExtension is used to specify the filename extensions that should be used to filter files in the reverse engineering file selection dialog. This property is used in conjunction with the property ImplementationExtension.

You can specify a number of extensions. They should be entered as a comma-separated list.

Default = h,inl

MFC

The MFC metaclass contains a property that affects the MFC type library.

DataTypes

The DataTypes property specifies classes to be modeled as MFC data types. There is only one predefined library (MFC) that contains only one class (Cstring). You can, however, expand this short list of classes by the addition of classes in this property or the creation of new libraries in the property files factory.prpfactory and site.prpsite.

Default = Cstring

MSVC60

The MSVC60 metaclass contains properties used to control the Microsoft Visual C++ environment.

Defined

The Defined property specifies symbols that are defined for the Microsoft Visual C++ version 6.0 (MSVC60) preprocessor. These symbols are automatically filled into the Name list of the Preprocessing tab of the Reverse Engineering Options window when you select Add > Dialect: MSVC60.

The default value is as follows:

__STDC__,__STDC_VERSION__,__cplusplus,__DATE__,
__TIME__,_WIN32,_cdecl,__cdecl,__int64=int,__stdcall,
__export,_export,_AFX_PORTABLE,_M_IX86=500,__declspec,
__MSC_VER=1200,__inline=inline,__far,__near,_far,_near,
__pascal,_pascal,__asm,__finally=catch,__based,
__inline=inline,__single_inheritance,__cdecl,__int8=int,
__stdcall,__declspec,__int16=int,__int32=int,__try=try,
__int64=int,__virtual_inheritance,__except=catch, __leave=catch,__fastcall,__multiple_inheritance)

IncludePath

The IncludePath property specifies necessary include paths for the Microsoft Visual C++ preprocessor. It is possible to specify the path to the site installation of the compiler as part of the site.prp, thus doing it only once and not for every project.

Default = empty string

Undefined

The Undefined property specifies symbols that must be undefined for the Microsoft Visual C++ preprocessor.

Default = empty string

Parser

The metaclass Parser contains properties that can be used to modify the way the parser handles code during reverse engineering.

AdditionalKeywords

The property AdditionalKeywords can be used to list non-standard keywords that may appear in the code that you reverse engineer. This allows Rhapsody to parse this code correctly during reverse engineering.

The value of this property should be a comma-separated list of the additional keywords you want to include.

Note that keywords with parameters are not supported, nor are keywords that consist of more than one word.

This property corresponds to the keywords listed on the Preprocessing tab of the Reverse Engineering Options window. Note that when you add additional keywords using the controls on the Preprocessing tab, these keywords are included in the value of the AdditionalKeywords property at the level of the active configuration.

Default = far,near

Defined

The Defined property specifies symbols and macros to be defined using #define. For example, you can enter the following to define name> as text with the appropriate intermediate character: /D name{=|#}text

Default = empty string

Dialects

The Dialects property specifies which symbols are added to the Preprocessing tab of the Reverse Engineering dialog box when that dialect is selected. The default value is MSVC60, which is itself defined by a metaclass of the same name under subject C_ReverseEngineering. This dialect specifies the symbols that are defined for the Microsoft Visual C++ environment. You can define your own dialect (in the site.prp file) and select it in the Dialects property. The default value for C is an empty string.

ForceExpansionMacros

By default, Rational Rhapsody reverse engineers macros such that when the code is regenerated, the macro definition and macro calls are generated as they appeared in the original code that was reverse engineered. (This behavior can be controlled with the property C_ReverseEngineering::ImplementationTrait::MacroExpansion.)

In some cases, you may find that you are not satisfied with the way that Rational Rhapsody imports the macro. For such situations, you can use the property ForceExpansionMacros to list specific macros that should be expanded during reverse engineering even if the value of the property MacroExpansion is set to False.

The value of this property should be a comma-separated list of the macros that you would like Rhapsody to expand during reverse engineering.

Default = Blank

IncludePath

The Preprocessing tab of the Reverse Engineering Options dialog allows you to specify an include path (classpath for Java) for the parser to use. The property IncludePath represents this path.

For the value of this property, you can enter a comma-separated list of directories. Note that you have to

specify subdirectories individually.

The directories you list here is combined with the directories specified in #include statements in order to find the necessary files. For example, if you have c:\d1\d2\d3\file.h, you can enter c:\d1\d2 as the value of this property and then use d3\file.h in the #include statement.

You should take into account that the value of this property also determines the structure of the source file directory when code is generated from the model. So, in the above example, the top-level directory created is d3.

Default = Blank

Undefined

The Undefined property specifies symbols and macros to be undefined using #undef.

Default = empty string

Promotions

The metaclass Promotion contains a number of properties used to specify whether Rational Rhapsody should add various advanced modeling constructs to your model based on relationships/patterns uncovered during reverse engineering.

EnableAttributeToRelation

The property EnableAttributeToRelation is used to specify whether Rational Rhapsody should add Associations to the model for attributes whose type is another class in the model.

For example, if you have two classes, A and B, and B contains an attribute of type A, Rational Rhapsody will add an Association to the model reflecting this relationship.

Default = Checked

EnableFunctionToObjectBasedOperation

The EnableFunctionToObjectBasedOperation property specifies whether object-based promotion is enabled during reverse engineering. Object-based promotion “promotes” a global function to comply with the pattern specified in the properties C_CG::Operation::PublicName and ProtectedName to be an operation of the class (object_type) defined in the function’s me parameter.

Default = Cleared

EnableResolveIncompleteClasses

Sometimes, during reverse engineering, Rational Rhapsody is not able to find the base class for a given class. The property EnableResolveIncompleteClasses is used to specify that if Rhapsody finds a class with the same name as the base class in a different location, it should assume that this class is the missing base class.

Default = Checked

Update

The metaclass Update contains properties used to control various aspects of the Rational Rhapsody behavior during and after reverse engineering.

CreateFlowcharts

The property CreateFlowcharts is used to specify whether or not Rational Rhapsody should automatically create flowcharts for operations during reverse engineering of code.

This property corresponds to the Create Flowcharts option on the Model Updating tab of the Reverse Engineering options window. Note that when you select the Create Flowcharts option, the value of the property CreateFlowcharts is modified at the level of the active configuration.

This property can be used in conjunction with the properties FlowchartCreationCriterion, FlowchartMinLOC, FlowchartMaxLOC, FlowchartMinControlStructures and FlowchartMaxControlStructures so that flowcharts are created only for operations that are within a given range in terms of lines of code or in terms of the number of control structures in the operation.

Default = Cleared

FlowchartCreationCriterion

If you have selected the option of having Rhapsody create flowcharts during reverse engineering, you can use the property FlowchartCreationCriterion to select the criterion that should be used to decide what operations Rhapsody should create flowcharts for.

The property corresponds to the radio buttons on the Model Updating tab of the Reverse Engineering options window.

The property can take the following values:

- Control Structures - the decision whether or not to generate a flowchart for an operation is based on the number of control structures in the operation. When this option is selected, the minimum and maximum number of control structures used to define the inclusion criterion are taken from the properties FlowchartMinControlStructures and FlowchartMaxControlStructures.
- LOC - the decision whether or not to generate a flowchart for an operation is based on the number of lines of code in the operation. When this option is selected, the minimum and maximum lines of code used to define the inclusion criterion are taken from the properties FlowchartMinLOC and FlowchartMaxLOC.

Default = LOC

FlowchartMaxControlStructures

If you have selected the option of having Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to Control Structures, then the property FlowchartMaxControlStructures is used to specify the maximum number of control structures that an operation can have, above which Rhapsody will not create a flowchart for it.

The property corresponds to the maximum control structures text box on the Model Updating tab of the Reverse Engineering options window.

Default = 10

FlowchartMaxLOC

If you have selected the option of having Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to LOC, then the property FlowchartMaxLOC is used to specify the maximum number of lines of code that an operation can have, above which Rhapsody will not create a flowchart for it.

The property corresponds to the maximum lines of code text box on the Model Updating tab of the Reverse Engineering options window.

Default = 100

FlowchartMinControlStructures

If you have selected the option of having Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to Control Structures, then the property FlowchartMinControlStructures is used to specify the minimum number of control structures that an operation must have in order to have Rhapsody create a flowchart for it.

The property corresponds to the minimum control structures text box on the Model Updating tab of the Reverse Engineering options window.

Default = 2

FlowchartMinLOC

If you have selected the option of having Rhapsody create flowcharts during reverse engineering, and you have set the value of the property FlowchartCreationCriterion to LOC, then the property FlowchartMinLOC is used to specify the minimum number of lines of code that an operation must have in order to have Rhapsody create a flowchart for it.

The property corresponds to the minimum lines of code text box on the Model Updating tab of the Reverse Engineering options window.

Default = 10

C_Roundtrip

The C_Roundtrip subject contains properties that affect roundtripping.

General

The General metaclass contains properties that control how changes to code are roundtripped in Rational Rhapsody.

NotifyOnInvalidatedModel

The NotifyOnInvalidatedModel property is a Boolean value that determines whether a warning window is displayed during roundtrip. This warning is displayed when information might get lost because the model was changed between the last code generation and the roundtrip operation.

(Default = Checked)

ParserErrors

The ParserErrors property specifies the behavior of roundtrip when a parser error is encountered. The possible values are as follows:

- Abort - Abort roundtrip whenever there is a parser error in the code. No changes is applied to the model.
- AbortOnCritical - Abort roundtrip if any critical parser errors are encountered in the code.
- AskUser - When Rhapsody encounters an error, it asks what you want to do.
- Ignore - Continue roundtrip, ignoring any parser errors that are encountered.

(C Default = AskUser)

PredefineIncludes

The PredefineIncludes property specifies the predefined include path for roundtripping.

(Default = empty string)

PredefineMacros

The PredefineMacros property specifies the predefined macros for roundtripping. The C default value is as follows:

OMADD_SER(p\, cvrtFunc), OMADD_OMSER(theEvent\, p), OMADD_ASER(p\,size\,sizeOfP\,cvrtF),

OMADD_ARCSER(p\size), OMADD_OMUNSER(t\,p\,destrFunc), OMADD_UNSER(t\,p\,destrFunc),
RICBAD_PARAM(p), BAD_MISSING_PARAM(p), OMDefaultThread=0, NULL=0,
OMDECLARE_GUARDED, RIC_EMPTY_STRUCT, OM_INSTRUMENT_OBJECT(theClass\,
thePackage\, thePackage\, isSingleton\, serVtbl),
OM_INSTRUMENT_PACKAGE(thePackage\,thePackage\,serVtbl),
OM_INSTRUMENT_CLASS(theClass\,thePackage\,theFullPackage\,isSingleton\,serVtbl),
OM_INSTRUMENT_OBJECT_TYPE(theClass\,thePackage\,theFullPackage\,isSingleton\,serVtbl),
OM_INSTRUMENT_FILE_CLASS(theClass\,theFullClassName\,thePackage\,theFullPackage\,isSingleton\,serVtbl),
OM_INSTRUMENT_FILE_OBJECT(theClass\,theFullClassName\,thePackage\,theFullPackage\,isSingleton\,serVtbl),
OM_INSTRUMENT_INSTANCE(me\,meAsReactive\,theClass),
OM_INSTRUMENT_EVENT_NO_UNSERIALIZE(theEvent\,thePackage\,theFullPackage\,signature),
OM_INSTRUMENT_EVENT_INSTANCE(rawMe\,theEventClass),
OM_INSTRUMENT_EVENT(theEventClass\, thePackage\, thePackage\, theEventClass),
RIC_DECLARE_MEMORY_ALLOCATOR_MEMBER(CLASSNAME),
RIC_DECLARE_MEMORY_ALLOCATOR(CLASSNAME),
RIC_MEMORY_ALLOCATOR_GET(CLASSNAME),
RIC_MEMORY_ALLOCATOR_RETURN(ME\, CLASSNAME),
RIC_IMPLEMENT_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM\,INCREMENTSIZE\,ISPROTECTED)

ReportChanges

The ReportChanges property defines which changes are reported (and displayed) by the roundtrip operation. The possible values are as follows:

- None - No changes are displayed in the output window.
- AddRemove - Only the elements added to, or removed from, the model are displayed in the output window.
- UpdateFailures - Only unsuccessful changes to the model are displayed in the output window.
- All - All changes to the model are displayed in the output window.

(Default = AddRemove)

RestrictedMode

The RestrictedMode property is a Boolean value that specifies whether restricted-mode roundtripping is enabled. This property can be modified on the configuration level. Restricted mode of full roundtrip enables you to roundtrip unusual usage of Rational Rhapsody elements, such as a class declaration in a user-defined type.

Restricted mode has more limitations, but preserves the model from unexpected changes. The additional limitations for restricted mode are as follows:

- User-defined types cannot be removed or changed on roundtrip because Rational Rhapsody code generation adds the Ignore annotation for a user-defined type declaration.
- Relations cannot be removed or changed on roundtrip.
- New classes are not added to the model.

(Default = Cleared)

RoundtripPreprocessorDirectives

By default, the Rational Rhapsody roundtripping feature takes into account changes made to preprocessor directives. The property RoundtripPreprocessorDirectives can be used to turn off roundtripping for the following types of preprocessor directives:

- elif
- else
- endif
- error
- if
- ifdef
- ifndef
- import
- line
- pragma
- undef
- using

Default = Checked

RoundtripScheme

Determines what type of changes can be roundtripped back into the model. The possible values are Basic and Advanced.

When set to Basic, only changes to the bodies of operations and actions are roundtripped into the model.

When set to Advanced, roundtripping also takes into account elements that have been added, such as attributes and operations, and can optionally take into account elements that have been modified or removed.

When set to Respect, roundtripping also takes into account the changes that are covered by the Rational Rhapsody "code respect" feature, for example, the order of class members.

Default = Respect

Update

The Update metaclass contains a property that controls the update process used during roundtripping.

AcceptChanges

The AcceptChanges property is an enumerated type that specifies which changes are applied to each CG element (attribute, operation, type, class, or package).

You can apply separate properties to each type of CG element. The possible values are as follows:

- All - All the changes can be applied to the model element.
- Default—1) Rhapsody will not roundtrip deletions if the updated code results in parser errors. 2) Rhapsody will not roundtrip the deletion of classes.
- NoDelete - All the changes except deletion can be applied to the model element. This setting prevents accidental removal of operations, constructors, attributes, relations, variables, instances, and functions.
- AddOnly - Apply only the addition of an aggregate to the model element. You cannot delete or change elements.
- NoChanges - Do not apply any changes to the model element.

Note that the value of the property is propagated to all the aggregates of an element. Therefore, if a package has the property value NoChanges, no elements in that package is changed.

Default = "Default" (in code-centric settings, default value is All)

DeploymentDiagram

The DeploymentDiagram subject contains the following metaclasses with properties for controlling the deployment diagram editor:

- AutoPopulate
- Communication_Path
- Complete
- ComponentInstance
- Depends
- DeploymentDiagramGE
- Flow
- NodeProcessor

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Bottom-Top

Comment

The Comment metaclass contains properties that control the appearance of comments in deployment diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Communication_Path

The Communication_Path metaclass contains properties that control the attributes of the communication path of the deployment diagram.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 255,0,0)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 255,0,0)

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

ComponentInstance

The ComponentInstance metaclass contains properties that control the attributes of the component instance.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,0,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,255)

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Constraint

The Constraint metaclass contains properties that control the appearance of constraints in deployment diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

Depends

The Depends metaclass contains a property that controls the appearance of dependency relation lines in deployment diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,0,255)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,255)

DeploymentDiagramGE

The DeploymentDiagramGE metaclass contains a property that controls the fill color used in deployment diagrams.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

Flow

The Flow metaclass contains properties that control how information flows are displayed in deployment diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,147,0)

flowKeyword

The flowKeyword property is a Boolean value that specifies whether the flow keyword for the information flow is displayed in the diagram.

Default = Checked

infoItemsColor

The infoItemsColor property specifies the color used to draw information items in diagrams. (Default = 0,0,255)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.

- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

NodeProcessor

The NodeProcessor metaclass contains properties that control the attributes of the component instance.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,128,0)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,128,0)

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Note

The Note metaclass contains properties that control the appearance of notes in deployment diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,128,64)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,128,255)

Requirement

The Requirement metaclass contains properties that control the appearance of requirements in deployment diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box.

(Default = 128,128,0)

Fillcolor

The Fillcolor property specifies the default fill color for the object. (Default = 0,255,255)

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 0,0,0)

DiagramPrintSettings

The DiagramPrintSettings subject contains properties that affect how diagrams are printed. It contains a single metaclass: General.

General

The General metaclass contains properties that control the default print settings for diagrams.

Footer

The Footer property specifies a string footer that is added to the bottom of the page for printed diagrams.

(Default = Page $PgNum of $Pages)

Header

The Header property specifies a string header that is added to the top of the page for printed diagrams.

(Default = $Name)

Orientation

The Orientation property specifies whether to print the diagram in portrait or landscape mode.

(Default = Portrait)

PrintBackground

The PrintBackground property specifies whether to print the background color for the diagram.

(Default = Cleared)

Scale

The Scale property specifies the default scaling factor to use when printing diagrams.

(Default = 100)

ShrinkToFitOnPage

The ShrinkToFitOnPage property specifies whether to scale the diagram as necessary so the entire diagram fits on a single page.

(Default = Checked)

Dialog

The Dialog subject contains properties that affect which properties are displayed in the Properties tab. The metaclasses are as follows:

- Attribute
- Dialog
- Component
- Configuration
- Dependency
- Diagrams
- Event
- File
- General
- ObjectModelDiagram
- Operation
- Package
- Project
- Relation
- SequenceDiagram
- Type
- UseCaseDiagram

All

Contains properties that affect the behavior of the Rational Rhapsody GUI when you select the View All property filter.

UserDefinedSubjects

The property UserDefinedSubjects allows you to enter a comma-separated list of subjects that you always want to be visible when the View All property filter is selected, regardless of the context.

Default = Blank

Attribute

The Attribute metaclass contains properties that control which subjects and metaclasses are displayed for attributes when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::Attribute::Animate, WebComponents::Attribute::WebManaged)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Attribute::AccessorGenerate, C_CG::Attribute::MutatorGenerate, C_CG::Attribute::VariableInitializationFile)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Attribute::AccessorGenerate, CPP_CG::Attribute::MutatorGenerate, CPP_CG::Attribute::ReferenceImplementationPattern)

JAVA_CommonProperties

The JAVA_CommonProperties property specifies which Rational Rhapsody Developer for Java-specific properties are displayed when you select the Common filter for the properties. (Default = JAVA_CG::Attribute::AccessorGenerate, JAVA_CG::Attribute::MutatorGenerate)

Class

The Class metaclass contains properties that control which subjects and metaclasses are displayed for classes when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::Class::UseAsExternal,CG::Class::Animate, CG::Class::ActiveThreadName,CG::Class::ActiveThreadPriority CG::Class::ActiveStackSize, CG::Class::ActiveMessageQueueSize, WebComponents::Class::WebManaged)

Ada_CommonProperties

The Ada_CommonProperties property specifies which Rational Rhapsody Developer for Ada-specific properties are displayed when you select the Common filter for the properties. (Default = Ada_CG::Class::Visibility,Ada_CG::Class::TaskBody)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Class::EnableDynamicAllocation, C_CG::Class::EnableUseFromCPP, C_CG::Class::GenerateDestructor,C_CG::Class::ImpIncludes, C_CG::Class::SpecIncludes, C_CG::Class::ObjectTypeAsSingleton)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Class::DeclarationModifier, CPP_CG::Class::Embeddable,CPP_CG::Class::ImpIncludes, CPP_CG::Class::IsReactiveInterface, CPP_CG::Class::SpecIncludes)

ClassifierRole

The ClassifierRole metaclass contains properties that control which subjects and metaclasses are displayed for classifier roles when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = Animation::ClassifierRole::MappingPolicy, Animation::ClassifierRole::DisplayMessagesToSelf)

Component

The Component metaclass contains a property that controls which subjects and metaclasses are displayed for components when you use the Common filter for the properties.

Ada_CommonProperties

The Ada_CommonProperties property specifies which Rational Rhapsody Developer for Ada-specific properties are displayed when you select the Common filter for the properties. (Default = Ada_CG::Component::AdaVersion)

Configuration

The Configuration metaclass contains properties that control which subjects and metaclasses are displayed for configurations when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::CGGeneral::GeneratedCodeInBrowser)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Configuration::ClassStateDeclaration, C_CG::Configuration::DefaultImplementationDirectory, C_CG::Configuration::DefaultSpecificationDirectory, C_CG::Configuration::InitializeEmbeddableObjectsByValue)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Configuration::ContainerSet, CPP_CG::Configuration::DefaultImplementationDirectory, CPP_CG::Configuration::DefaultSpecificationDirectory, CPP_CG::Configuration::InitializeEmbeddableObjectsByValue)

Dependency

The Dependency metaclass contains properties that control which subjects and metaclasses are displayed for dependencies when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::Dependency::UsageType)

Ada_CommonProperties

The Ada_CommonProperties property specifies which Rational Rhapsody Developer for Ada-specific properties are displayed when you select the Common filter for the properties. (Default =

Ada_CG::Dependency::CreateUseStatement)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Dependency::UseNameSpace)

Diagrams

The Diagrams metaclass contains a property that controls which subjects and metaclasses are displayed for diagrams when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = General::Graphics::ShowLabels)

Event

The Event metaclass contains properties that control which subjects and metaclasses are displayed for events when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::Event::Animate, CG::Event::DeleteAfterConsumption, WebComponents::Event::WebManaged)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Event::EnableDynamicAllocation)

File

The File metaclass contains properties that control which subjects and metaclasses are displayed for files when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = WebComponents::File::WebManaged)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::File::ImplementationHeader, C_CG::File::SpecificationHeader)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::File::ImplementationHeader, CPP_CG::File::SpecificationHeader)

JAVA_CommonProperties

The JAVA_CommonProperties property specifies which Rational Rhapsody Developer for Java-specific properties are displayed when you select the Common filter for the properties. (Default = JAVA_CG::File::Header)

General

The General metaclass contains a property that controls which filter is applied to the list of properties.

PropertiesDialogDefaultFilter

The PropertiesDialogDefaultFilter property specifies the default filter used for the list of properties. The possible values are as follows:

- All - Displays all the available properties, according to context. This is the default view for projects created before Version 4.1.
- Common - Displays the properties contained in the Dialog::metaclass::Common property. This is the default view for Version 4.1 projects.
- Overridden - Displays only those properties whose default values have been overridden, up to the project level.
- When you select this view, the GUI displays all the overridden properties from the selected element up

to the scope of the project; overridden properties at a scope higher than the selected element are grayed out.

- Locally Overridden - Displays only the locally overridden properties for the selected element. A selected element is a project, component, configuration, package, diagram, view element, and any other model element displayed in the browser.

(Default = Common)

ObjectModelDiagram

The ObjectModelDiagram metaclass contains a property that controls which subjects and metaclasses are displayed for OMDs when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = General::Graphics::ShowLabels, ObjectModelGe::Class::ShowAttributes, ObjectModelGe::Class::ShowOperations, ObjectModelGe::Class::ShowName, ObjectModelGe::Class::ShowStereotype, ObjectModelGe::Complete::Complete_Relation, ObjectModelGe::Package::ShowName)

Operation

The Operation metaclass contains properties that control which subjects and metaclasses are displayed for operations when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::Operation::Animate,CG::Operation::Concurrency, CG::Operation::VariableLengthArgumentList, WebComponents::Operation::WebManaged)

Ada_CommonProperties

The Ada_CommonProperties property specifies which Rational Rhapsody Developer for Ada-specific properties are displayed when you select the Common filter for the properties. (Default = Ada_CG::Operation::EntryCondition, Ada_CG::Operation::LocalVariablesDeclaration, Ada_CG::Operation::Renames)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Operation::ImplementationEpilog, CPP_CG::Operation::ImplementationProlog, CPP_CG::Operation::SpecificationEpilog, CPP_CG::Operation::SpecificationProlog, CPP_CG::Operation::ThrowExceptions)

JAVA_CommonProperties

The JAVA_CommonProperties property specifies which Rational Rhapsody Developer for Java-specific properties are displayed when you select the Common filter for the properties. (Default = JAVA_CG::Operation::IsNative, JAVA_CG::Operation::ThrowExceptions)

UseReturnTypeFromCG

The property UseReturnTypeFromCG specifies whether the signature field on the General tab of the features dialog for operations should display the actual return type that is generated during code generation.

Default = Cleared

Package

The Package metaclass contains properties that control which subjects and metaclasses are displayed for packages when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = General::Graphics::ShowLabels, SequenceDiagram::General::RealizeMessages, SequenceDiagram::General::ShowSequenceNumbers, SequenceDiagram::General::ClassCentricMode, SequenceDiagram::General::ShowArguments, ObjectModelGe::Class::ShowAttributes, ObjectModelGe::Class::ShowOperations, ObjectModelGe::Class::ShowName)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Package::ImpIncludes,C_CG::Package::SpecIncludes)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific

properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Package::Animate,CPP_CG::Package::SpecIncludes, CPP_CG::Package::ImpIncludes, CPP_CG::Package::DefineNameSpace, CPP_CG::Package::ImpIncludes)

Project

The Project metaclass contains properties that control which subjects and metaclasses are displayed for projects when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = General::Graphics::MaintainWindowContent, General::Graphics::DragOnContourOnly, General::Graphics::grid_display, General::Graphics::grid_snap, SequenceDiagram::General::RealizeMessages, SequenceDiagram::General::ShowSequenceNumbers, SequenceDiagram::General::CleanupRealized, SequenceDiagram::General::ClassCentricMode, SequenceDiagram::General::ShowArguments, ObjectModelGe::Class::ShowAttributes, ObjectModelGe::Class::ShowOperations, ObjectModelGe::Class::ShowName, ConfigurationManagement::General::CMTool, ConfigurationManagement::General::UseSCCtool, RTInterface::DOORS::InstallationDir, RTInterface::DOORS::LmLicenseFile, CG::CGGeneral::GeneratedCodeInBrowser)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Configuration::InitializeEmbeddableObjectsByValue, C_CG::Attribute::AccessorGenerate, C_CG::Attribute::MutatorGenerate)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Attribute::AccessorGenerate, CPP_CG::Attribute::MutatorGenerate, CPP_CG::Configuration::InitializeEmbeddableObjectsByValue)

JAVA_CommonProperties

The JAVA_CommonProperties property specifies which Rational Rhapsody Developer for Java-specific properties are displayed when you select the Common filter for the properties. (Default = JAVA_CG::Attribute::AccessorGenerate, JAVA_CG::Attribute::MutatorGenerate)

Relation

The Relation metaclass contains properties that control which subjects and metaclasses are displayed for relations when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = CG::Relation::Ordered)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Relation::ImplementWithStaticArray)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Relation::ImplementWithStaticArray, CPP_CG::Relation::Static)

JAVA_CommonProperties

The JAVA_CommonProperties property specifies which Rational Rhapsody Developer for Java-specific properties are displayed when you select the Common filter for the properties. (Default = JAVA_CG::Relation::Static)

SequenceDiagram

The SequenceDiagram metaclass contains a property that controls which subjects and metaclasses are displayed for sequence diagrams when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = General::Graphics::ShowLabels, SequenceDiagram::General::RealizeMessages, SequenceDiagram::General::ShowSequenceNumbers, SequenceDiagram::General::ShowArguments, Animation::ClassifierRole::MappingPolicy, Animation::ClassifierRole::DisplaysMessagesToSelf)

Stereotype

The Stereotype metaclass contains a property that controls which subjects and metaclasses are displayed for stereotypes when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = Model::Stereotype::BrowserIcon,Model::Stereotype::BrowserGroupIcon,Model::Stereotype::DrawingShape,Model::Stereot

Type

The Type metaclass contains properties that control which subjects and metaclasses are displayed for user-defined types when you use the Common filter for the properties.

Ada_CommonProperties

The Ada_CommonProperties property specifies which Rational Rhapsody Developer for Ada-specific properties are displayed when you select the Common filter for the properties. (Default = Ada_CG::Type::Visibility, Ada_CG::Type::DeclarationPosition)

C_CommonProperties

The C_CommonProperties property specifies which Rational Rhapsody Developer for C-specific properties are displayed when you select the Common filter for the properties. (Default = C_CG::Type::AnimSerializeOperation, C_CG::Type::AnimUnserializeOperation)

CPP_CommonProperties

The CPP_CommonProperties property specifies which Rational Rhapsody Developer for C++-specific properties are displayed when you select the Common filter for the properties. (Default = CPP_CG::Type::AnimSerializeOperation, CPP_CG::Type::AnimUnserializeOperation)

UseCaseDiagram

The UseCaseDiagram metaclass contains a property that controls which subjects and metaclasses are displayed for UCDs when you use the Common filter for the properties.

CommonProperties

The CommonProperties property specifies which properties are displayed when you select the Common filter for the properties. (Default = General::Graphics::ShowLabels, UseCaseGe::UseCase::ShowName, UseCaseGe::UseCase::ShowStereotype, UseCaseGe::Complete::Complete_Relation, UseCaseGe::Package::ShowName)

Eclipse

The Eclipse subject contains properties that affect which properties are displayed in the Properties tab.

The metaclasses are as follows.

- Configuration
- DefaultEnvironments

Configuration

The Configuration metaclass contains properties that control the display of the Rational Rhapsody browser.

InvokeExecutable

The InvokeExecutable property (under Eclipse::Configuration) points to the executable. Keywords:$executable - the IDE executable as read from Rhapsody.ini.

The possible values are as follows:

- Always - Rational Rhapsody displays a confirmation dialog each time you try to delete an item from the model.
- Never - Confirmation is not required to delete an element.
- WhenNeeded - Rational Rhapsody asks for confirmation if there are references to the element (or for some other reason).

(Default = $executable)

InvokeParameters

The InvokeParameters property (under Eclipse::Configuration) control the parameters for the command line.

Keywords:

$workspace as specified in the Rational Rhapsody Tags for the Eclipse Configuration.

$RhpClientPort: the port number that Rational Rhapsody uses to be a client to Eclipse, as specified using Rhapsody’s menu Code IDE options .

$RhpServerPort: the port number that Rational Rhapsody uses to be a server to Eclipse (Default = -data $workspace -vmargs -DRhpClientPort= $RhpClientPort -DRhpServerPort= $RhpServerPort)

DefaultEnvironments

A default Rhapsody environment is chosen according to the type of project that the user creates in IDE.

The following are examples of the types of projects that would affect the choice of the Rational Rhapsody environment:

- The user creates new Eclipse configuration in Rational Rhapsody.
- Workbench is brought forward and "Create new project" wizard is displayed.
- The user creates a Workbench Real Time Project in Workbench.
- Rhapsody is notified that an active Eclipse configuration is coupled with a Workbench project.

Eclipse

The Eclipse property (under DefaultEnvironments::Eclipse) is the default environment in the settings tab for generic Eclipse (CDT) projects.

(Default = Cygwin)

Workbench

The Workbench property (under Eclipse::Configuration) is the default environment in the settings tab for generic Eclipse (CDT) projects.

(Default = WorkbenchManaged)

WorkbenchKernel

The WorkbenchKernel property is set if the user creates a Doanloadable Kernel module project in Workbench and wants the default environment to be "WorkbenchManaged."

WorkbenchRTP

The WorkbenchRTP property is set if the "DefaultEnvironments" for a Workbench Real Time project is mapped to Eclipse:DefaultEnvironments:WorkbenchKernel property and its default value is Workbenchmanaged_RTP.

Export

The metaclass Export contains properties related to the exporting of Eclipse projects to create Rhapsody

models.

AssociateWithOriginalProjectOnExport

When you export an Eclipse project to Rational Rhapsody, the Export to Rational Rhapsody Model dialog contains an option "Associate Rhapsody model with original Eclipse project", which is selected by default. This means that after the contents of the Eclipse project are brought into a Rhapsody model, the resulting model is linked to the original Eclipse project, and when code is generated from the model, it is stored in the original Eclipse project.

In some cases, however, you may want to sever any connection between the original Eclipse project and the Rational Rhapsody model after the initial import into Rational Rhapsody. If you clear the "Associate Rhapsody model with original Eclipse project" check box, then the original Eclipse project is only used for the initial import into Rational Rhapsody. A dialog is opened where you can indicate that you want the code generated from the Rational Rhapsody model to be stored in a new Eclipse project, or select an existing Eclipse project that you would like to use to house the code generated from Rhapsody.

Default = Checked

General

The General subject controls the general aspects of the Rational Rhapsody display. It contains the following metaclasses:

- Model
- Graphics
- Profile
- Relations
- Report
- ReporterPLUS
- Workspace

Graphics

The Graphics metaclass contains properties that determine the general behavior of graphic editors, such as whether you can drag a graphic element by clicking on its bounding box.

AutoScrollMargin

The AutoScrollMargin controls how responsive the auto scrolling functionality is. The auto scroll begins scrolling when the mouse pointer enters the auto scroll margins, which are virtual margins that define a virtual region around the drawing area (starting from the window frame and going X number of points into the drawing area).

The AutoScrollMargin property defines the X number of points the margins enter into the drawing area. If you specify a large number for this property, the margin becomes bigger, thereby making the auto scroll more sensitive. Set this property to 0 to disable auto scrolling.

(Default = 50)

AutoScrollOnSelecting

Rhapsody includes an autoscroll feature that kicks in when you are selecting objects within a given region and you approach the edge of the diagram window. This allows you to extend your selection to include objects that are currently outside the viewable area of the window.

The property AutoScrollOnSelecting can be used to disable/enable this feature.

You can use the property AutoScrollMargin to control when the autoscrolling kicks in by changing the size of the region considered to be the edge of the viewable area.

Default = Checked

ClassBoxFont

The ClassBoxFont property specifies the default font for new class attributes and operations. To change the font used for the class itself, use the Format window. (Default = Arial 10 NoBold NoItalic)

CompartmentsTitleFont

When classes are displayed in Specification view, compartments are displayed for elements such as attributes and operations.

If you have used the Display Options dialog or the property ShowCompartmentsTitle to specify that headings should be displayed to identify the different compartments, you can use the property CompartmentsTitleFont to choose the font that Rational Rhapsody should use for these compartment headings.

When you click the ".." button next to the property value, a font chooser dialog is displayed.

Default = Arial 14 NoBold Italic

CRTerminator

The CRTerminator property specifies how multiline fields in notes and statechart names should interpret a carriage return (CR). Note that single-line fields, such as relation and role names and messages in sequence diagrams, always interpret a CR as a command to finish editing. The possible values are as follows:

- Checked - Multiline fields interpret a CR as a command to finish editing. Use Ctrl+CR to insert a new line.
- Cleared - Multiline fields interpret a CR as a new line. Use Ctrl+CR to exit from Edit mode.

(Default = Cleared)

DefaultBoxView

The property DefaultBoxView determines how new classes and objects are displayed in object model diagrams - "Specification" view (with compartments) or "Structured" view (without compartments).

The property can take any of the following values:

- Specification - When you create a class/object on an object model diagram, or drag a class/object from the browser, it is opened on the diagram using Specification view. Also, if you select a class and select "Make an Object" from the context menu, the object created is opened using Specification view (regardless of the view that was used previously for the class it is based on).
- Structured - When you create a class/object on an object model diagram, or drag a class/object from the browser, it is opened on the diagram using Structured view. Also, if you select a class and select "Make an Object" from the context menu, the object created is opened using Structured view (regardless of the view that was used previously for the class it is based on).

- Default - When you create a class/object on an object model diagram, or drag a class/object from the browser, it is opened on the diagram using Specification view. If you select a class, and select "Make an Object" from the context menu, the object created is opened using the view that was used previously for the class it is based on.

Default = Default

DeleteConfirmation

The DeleteConfirmation property specifies whether confirmation is required before deleting a graphical element from the model. Note that this property does not apply to statechart elements, which have a separate DeleteConfirmation property.

The possible values are as follows:

- Always - Rational Rhapsody displays a confirmation dialog each time you try to delete an item from the model.
- Never - Confirmation is not required to delete an element.
- WhenNeeded - Rational Rhapsody asks for confirmation if there are references to the element (or for some other reason).

(Default = Always)

DiagramOriginPolicy

The DiagramOriginPolicy property defines the diagram origin policy.

- ZeroBased - The top left corner of the diagram is fixed and set to coordinates (0,0)
- ByComponentsBounds - The origin of the diagram is dynamic and is set by the diagram content

(Default = ByComponentsBounds)

DragOnContourOnly

The DragOnCountourOnly property specifies the move-by-drag policy in the diagrams. If this property is Checked, an element can by moved around the diagram only by clicking and dragging its contour rather than just grabbing somewhere around its bounds.

(Default = Checked)

EnableImageView

The EnableImageView property specifies whether images associated with graphical elements are displayed by default (Checked) instead of the element’s standard geometric shape.

This property has a user interface component in the element Display Options window.

(Default = Cleared)

ExportedDiagramScale

This property specifies how an exported diagram is scaled and whether it can be split into separate pages for better readability. The possible values are as follows:

- FitToOnePage - Scale the exported diagram as necessary so it can fit one one page.
- NoPagination - Export the diagram as a whole, at 100% scaling. This option is for users who use HTML viewers.
- UsePrintLayout - Export the diagram using same the same settings as specified in the diagram print settings. In essence, the exported diagram is the same as if it were printed.
- For example, if the diagram print scale is 200% and the orientation is Landscape, the diagram is exported in the same way. The diagram is split as per the print page bounds (dashed lines) shown on the diagram.

In previous versions, this property included a zoom percentage (40 to 100, 150, and 200), which zoomed the diagram to the specified percentage during export. These values have been removed in Version 6.0. However, if you previously used these values in your model, they will still work.

(Default = FitToOnePage)

FitBoxToItsTextuals

This property specifies whether to resize boxes automatically to fit their text content (such as names, attributes, or operations). (Default = Checked)

FixedConnectionPoints

When you attach a connector to a diagram element, Rational Rhapsody treats the connection point as a flexible connection point. If you move a connected element, Rational Rhapsody may change the position of the connection point if it will improve the appearance of the diagram.

The property FixedConnectionPoints makes it possible to create connection points that are fixed and will not be adjusted by Rhapsody when elements are moved. When this property is set to True, if you draw a connector to the edge of a diagram element, the connection point is a fixed point. If you attach the connector to the middle of an element when drawing the connector, it is a flexible connection point.

Default = Cleared

FlickerFree

This property is currently unused.

grid_color

The grid_color property specifies the default color used for the grid lines. (Default = 0,0,0)

grid_display

The grid_display property currently has no effect.

grid_snap

The grid_snap property specifies whether the Snap to Grid feature is enabled for new diagrams, regardless of whether the grid is actually displayed. The possible values are as follows:

- Checked - Objects are forced to align with the grid when you draw, move, or stretch them.
- Cleared - Objects are not forced to align with the grid when you draw, move, or stretch them.

(Default = Cleared)

grid_spacing_horizontal

The grid_spacing_horizontal property specifies the spacing, in world coordinates, between grid lines along the X-axis when the grid is enabled for diagrams Note that you can set this value in the GUI by selecting Layout Grid Grid Properties and changing the value of the field Grid Spacing, Horizontal.

(Default = 0.125)

grid_spacing_vertical

The grid_spacing_vertical property specifies the spacing, in world coordinates, between grid lines along the Y-axis when the grid is enabled for diagrams Note that you can set this value in the GUI by selecting Layout Grid Grid Properties and changing the value of the field Grid Spacing, Vertical.

(Default = 0.125)

HighlightSelection

The HighlightSelection property specifies whether items should be highlighted when you move the cursor over them in a diagram. (Default = Checked)

ImageViewLayout

Specifies how to show as associated image. The user can select from three different options:

- ImageOnly - Displays only a large image in the diagram.
- Structured - Displays the image in addition to the top of the diagram structure.
- Compartment - Displays the image in addition to the entire diagram structure.

(Default = ImageOnly)

MaintainWindowContent

The MaintainWindowContent property specifies whether the viewport (the part of a diagram displayed in the window) is kept for window resizing operations when you change the zoom level, providing additional space in the diagram in a smooth manner. The possible values of the property are as follows:

- Checked - The elements are scaled according to the zoom factor so you see the same elements in the window, regardless of scaling.
- Cleared - As the diagram is scaled, some elements are hidden or revealed, depending on the zoom. This is the behavior provided by previous versions of Rational Rhapsody.

The following operations change the window size:

- Maximize/restore
- Tile
- Cascade
- Manual resizing by dragging the edge of the window

You can also access this functionality in the GUI by selecting View Maintain Window Content. (Default = Cleared)

MarkMisplacedElements

The MarkMisplacedElements property specifies whether misplaced elements are marked in a special way. Previously, misplaced elements were shown with a small X in the upper corner.

In Rhapsody 6.0, misplaced elements are marked by default with red, cross-hatched lines in the diagram. However, you can change the marking used via the Format window (select the project, package, or diagram, select Format, then select the misplaced element).

A misplaced element is one that looks different ly in the diagram than its actual definition in the model. For example, in a diagram, it might look as though object A is contained by object B, although that is not how object A is actually defined in the model.

Therefore, object A would be marked as misplaced using red, cross-hatched lines.

(Default = Checked)

MultiScaleOneByOne

The MultiScaleOneByOne property specifies whether objects in the diagram are scaled as a group (Cleared) or each independently (Checked). (Default = Cleared)

PopulateClassSize

The property PopulateClassSize determines the size of the class/object when Rhapsody autopopulates an OMD. The possible values are:

- OldStyle - uses the class size that was used for autopopulate prior to version 6.1 MR-1: greater in width than in height
- DefaultStyle - uses the default class size that is used for creating ordinary OMDs: greater in heght than in width. When this option is selected, some or all of the attributes/operations is opened, depending on how many the class contains.

(Default = OldStyle)

PopulateExpandedSelection

The property PopulateExpandedSelection enables a larger selection ability in the populate window. If the checkbox is checked, when you right-click an item in the tree-list of the populate dialog is, a popup menu appears with the following options:

- Select Only
- Select with decedents
- Select with Base classes

(Default = Cleared)

PopulateHierarchyStyle

The property PopulateHierarchyStyle determines the Hierarchy layout style. Users can select one of two options:

- Top-Bottom: In the case of inheritence, the base class is at the top and the derived class at the bottom (Default style). This also applies to other links, like association, where the target class of the association is at the top, and the source class at the bottom
- Bottom-Top: Opposite of Top-Bottom, i.e. the base class is on the bottom and the derived class on the top (this was the previous layout style)

(Default = Top-Bottom)

PopulateMaxBoxSize

The property PopulateMaxBoxSize is used to limit the size of diagram elements when a diagram is auto-populated.

The value of the property is a comma-separated list of four integers. The maximum width in pixels is the difference between the third number and the first number. The maximum height in pixels is the difference between the fourth number and the second number.

For example, if you entered 0,0,200,200, the maximum size would be 200 pixels by 200 pixels.

If the value is 0,0,0,0 then no size limit is applied to elements during auto-population of diagrams - elements will be as large as necessary to accommodate the contained text.

Default = 0,0,0,0

PrintLayoutExportScale

The PrintLayoutExportScale property specifies the factor by which the Windows metafile format (WMF) files are scaled down in order to fit on one page. The default value, 75, guarantees that the diagram will fit into a single page in Microsoft Word.

RepeatedDrawing

The RepeatedDrawing property specifies whether repetitive drawing mode (stamp mode) is enabled. Repetitive drawing mode enables you to create a box element with a single click; double-clicking produces two of the same box elements.

By default, each time you want to add an element to a diagram, you must first click the appropriate icon in the drawing toolbar.

In some cases, however, you may want to add a number of elements of the same type. To facilitate this, Rational Rhapsody includes a "repetitive drawing mode."

To enter "repetitive drawing mode," click the "stamp" icon in the Layout toolbar. After selecting a tool in the drawing toolbar, you are able to continue drawing elements of that type without selecting the tool again each time.

If you choose a different tool from the toolbar, then Rational Rhapsody allows you to draw multiple elements of the newly selected type.

After you click the icon, Rational Rhapsody remains in "repetitive drawing mode" until you turn it off. To turn off the repetitive mode, just click the "stamp" icon a second time.

(Default = Cleared)

RotateDiagramOnExport

The property RotateDiagramsOnExport determines whether a diagram should be rotated on export. This property takes effect only when General::Graphics::ExportedDiagramScale is set to UsePrintLayout. The possible values are RotateLeft, RotateRight, and No (Default value).

This property replaced the property LandscapeRotateOnExport. If an older model overrode the property LandscapteRotateOnExport, it will override this property as well.

ScaleToFitExportedDiagram

The ScaleToFitExportedDiagram property specifies whether the diagram is scaled to fit the window before it is exported (as a metafile) to other applications, such as the COM API or Reporter Pro. (Default = Checked)

ShowActivityFrame

Select this property to add an activity frame automatically to a new activity diagram when it is created. (Default = Cleared)

ShowCompartmentsTitle

When classes are displayed in Specification view, compartments are displayed for elements such as attributes and operations.

The property ShowCompartmentsTitle can be used to specify that headings should be displayed to identify the different compartments.

The property can be set at the diagram level or higher.

When you change the value of the property, it affects the appearance of any classes subsequently added to the diagram, but does not affect the appearance of classes already on the diagram.

When this property is set to True, you can use the property CompartmentsTitleFont to choose the font that Rational Rhapsody should use for these compartment headings.

Default = Cleared

ShowDiagramFrame

This property controls whether diagram frames (the line border around diagrams) is shown by default or not. This feature can be overridden by right-clicking anywhere in the in the diagram canvas and selecting “Show/Hide Diagram Frame."

(Default = Cleared)

ShowEdgeTracking

The ShowEdgeTracking property specifies whether to show the “ghost” edges of a linked element when you move it. This is set to True by default, which means you can see the edges. (Default = Checked)

ShowLabels

The ShowLabels property is a Boolean value that specifies whether to display labels instead of names in the browser or diagrams, depending on which property is set.

(Default = Cleared)

ShowMultipleStereotypes

The property ShowStereotypes is a Boolean property in the browser. Setting this property to Cleared shows only the first stereotype of a certain element even if it has several stereotypes.

(Default = Checked)

ShowStereotypes

The property ShowStereotypes determines whether the browser displays the stereotype applied to a model element, alongside the name of the element. The possible values for this property are:

- No - stereotype is not displayed
- Prefix - stereotype is displayed to the left of the element name
- Suffix - stereotype is displayed to the right of the element name

The default value is Prefix. For projects created with Rational Rhapsody 6.0 or earlier, this property is overridden and set to No. The property is set at the project level. When the user selects View > Browser Display Options > Show Stereotype from the main menu, the property is assigned the value Prefix.

When the user deselects the Show Stereotype menu item, the property is assigned the value No.

StereotypeBitmapTransparentColor

The StereotypeBitmapTransparentColor property creates a “transparent” background for bitmaps associated with stereotypes (so only the graphics are displayed in the class box).

To create a transparent background, set this property to the RGB value of the bitmap’s background.

See the Rational Rhapsody Help for information on associating bitmaps with stereotypes.

(Default = 255, 0, 255)

TemplateParamsLayout

In version 7.2 of Rational Rhapsody, a change was made to the way that template parameters are displayed in object model diagrams.

Previously, if you changed the size of the template element in the diagram, the size of the box that displays the template parameters would also change proportionately.

Beginning with 7.2, the size of the box containing the template parameters does not change when you change the size of the template element in the diagram.

The backward compatibility profile, CGCompatibilityPre72Cpp, contains a property called TemplateParamsLayout, which is used to provide the previous Rhapsody behavior. The possible values for the property are:

- Regular - the size of the template parameter box changes together with the template element
- FixedSize - the size of the template parameter box remains a fixed size when you change the size of the template element

Default = Regular

Tool_tips

The Tool_tips property enables the display of tooltips.

(Default = Checked)

Model

The Model metaclass contains properties that control the general features of model elements, such as the format of element names.

ActiveCodeViewSensitivity

The ActiveCodeViewSensitivity property controls the update rate of the active code view (ACV). The possible values are as follows:

- ElementSelection - The ACV is updated whenever you modify the selection and whenever there are changes in the model.
- OnFocus - The ACV is updated only when you set it as the focused view (by clicking on the ACV view pane.

(Default = ElementSelection)

ActualCallRegExp

The ActualCallRegExp property specifies the regular expression describing the format of a legal actual call to an operation when the call is part of a transition. Usually, it is the action part of a transition. The default regular expression ^(.+)\(.*\)$ is evaluated as follows:

- The circumflex character (^) means that matching should begin from the start of the string. This matches a prefix of the string. The dollar sign ($) matches the NULL character at the end of the input string.
- The sequence “ (.+)\(“ means to match one or more characters until an open parenthesis is found. This implies that if there are no characters before the opening parenthesis in the input string, there is no match.
- The period matches any single character. For example, the expression “...” would match any three characters.
- The opening parenthesis (“ (“) has a special meaning in regular expressions. Therefore, it is preceded with the backslash escape character, which tells the parser to ignore the usual meaning of the opening parenthesis and look for a literal " (" character in the string. For example, to match the string " (a)", you would use the regular expression " \(a\)".

(Default = ^(.+)\(.\)$)*

AdditionalHelpersFiles

The property AdditionalHelpersFiles can be used to specify additional .hep files that should be loaded for a project, beyond the .hep file specified using the property HelpersFile.

The value of this property should be a comma-separated list of the additional .hep files you would like to associate with the project.

Default = Blank

AdditionalLanguageKeywords

The AdditionalLanguageKeywords property specifies a comma-separated list of language-specific keywords to be color-coded by the Rational Rhapsody internal editor, in addition to the default keywords (such as "class" and "public").

(Default = empty string)

AddNewMenuStructure

The AddNewMenuStructure property is used to define the structure of the Add New menu that appears when you right-click an item on the Rational Rhapsody browser.

The structure of the property is: Metaclass name,submenu name/Metaclass name,submenu name/Metaclass name, ... For example: The property: Class,rpy_seperator,Package,Annotations/Constraint,Annotation/rpy_seperator,Annotations/Requirement,Annotations/Comm will create the following "add new" menu: Class Package Annotations Constraint Requirement Comment

ApplyNewTermSemantic

The ApplyNewTermSemantic property applies to NewTerms. The NewTerm feature is used to define new types based on exisiting out-of-the-box types. Once done, a new type exists. If the ApplyNewTermSemantic property is checked, the NewTerms will work. If cleared, NewTerms will not work.

(Default = Checked)

AutoCascadeAddNewMenu

The AutoCascadeAddNewMenu property, automatically cascades the "add new" menu (checked) according to the profiles that contain each NewTerm. The NewTerm feature is used to define new types based on exisiting out-of-the-box types. Once done, a new type exists.

(Default = Checked)

AutoSaveInterval

The AutoSaveInterval property specifies the interval, in minutes, at which Rhapsody automatically saves your project. The following naming scheme applies:

- Project file config.rpy is copied to the file _auto.rpy.
- Repository \config_rpy is copied to the directory \config_auto_rpy.

(Default = 0)

AutoSynchronize

The AutoSynchronize property is a Boolean value that determines whether Rational Rhapsody will run synchronization.

When this property is Checked, each time Rhapsody gets the focus (for example, if you leave Rhapsody to read e-mail, then switch back to Rational Rhapsody), Rational Rhapsody will run the synchronize functionality.

The started synchronize can be a synchronization with the files on the file system, view, or CM archive, depending on the environment. See the Team Collaboration Guide for more information on configuration management tools.

(Default = Cleared)

AvailableMetaclasses

Rhapsody allows you to hide any out-of-the-box element types that your users will not need. The availability of metaclasses is determined by the property AvailableMetaclasses. This property is defined using a comma-separated list of strings.

To keep all of the out-of-the box metaclasses, leave this property blank.

To limit the availability of certain metaclasses, use this property to indicate only the metaclasses that you would like to have available. The strings to use to represent the different metaclasses can be found in the file metaclasses.txt in the Doc directory of your Rhapsody installation.

BackUps

The BackUps property specifies the maximum number of backups created when you save.

The possible values are None, One, and Two.

(Default = None)

BlockIsSavedUnit

The BlockIsSavedUnit property determines whether new blocks are saved as units (separate files) by default. (Default = Cleared)

CheckRoundtrip

The CheckRoundtrip property determines whether roundtrip is enabled after performing a Check Model operation. Sometimes, code generation unexpectedly displays a message saying that files have been externally changed and that roundtrip might be needed, even if this is not the case.

To disable roundtrip (and this message), set this property to Cleared. When roundtrip is enabled, a shortcut confirmation option lets you select a Yes to All or No to All button.

(Default = Checked)

ClassCodeEditor

The ClassCodeEditor property specifies which kind of editor is started when editing classes. The editor can be displayed in either a modal or modeless window. A modeless window enables you to do other work in other windows while it is open, whereas a modal window does not allow you to select any other window while it is open.

The possible values are as follows:

- Internal - Use the Rational Rhapsody internal editor for both modal and modeless editing.
- Associate - Use the editor associated with .h and .cpp files as set in the Windows registry for modeless editing, and use the internal editor for modal editing.
- CommandLine - Use the editor specified in the EditorCommandLine property for both modal and modeless editing.

(Default = Internal)

ClassIsSavedUnit

The ClassIsSavedUnit property determines whether new classes are saved as units (separate files) by default. (Default = Cleared)

CommonClassifiers

The property CommonClassifiers can be used to control what packages should be used for populating the Realization drop-down list on the General tab of the features dialog for instance lines in sequence diagrams. Use of this property can improve GUI response time for large models.

For the value of this property, enter a comma-separated list of package names, for example: pkg1,pkg2. (The list should not contain spaces.) Only the classifiers from the specified packages is included in the drop-down list.

If the property is left empty, classifiers from all the packages is included in the drop-down list.

If you choose to use this property to specify a list of packages, the drop-down list will include a "Select" option that will allow the user to select classifiers that are not displayed in the list by default.

CommonList

The CommonList property controls which elements appear in the top section of the Add New menu (referred to as the common list), when applicable. You can re-order, remove, or re-add any of these elements by doing so through the CommonList property.

Note the following:

- Whatever element that is removed from CommonList will appear in the middle portion of the Add New menu if that element is relevant for your project. The element must appear somewhere if it is a valid element.
- The General::Model::AddNewMenuStructure property overrides this property.

Default =
Function,Variable,Attribute,PrimitiveOperation,TriggeredOperation,Recption,Constructor,Event,Package,Component,Confi
secase,Requirement,

CommonTypes

The CommonTypes property specifies which types are listed as alternatives for attributes, variables, and arguments to make commonly used types easily accessible.

For example, you can omit the types listed in the Predefined package from the list of alternative types, and instead specify a list of packages that contain the types defined according to your design and code standards. Using the CommonTypes property, you can specify a list of packages and files that contain types that you use often, or types that are "basic" types for the project.

The value of the property is a comma-separated list of full paths of packages; the types are listed in the order specified in the property. If the value is $ALL, all types from all packages are included in the list.

If this property is empty, Rational Rhapsody uses Version 5.2 behavior. Note that if you do not explicitly include the PredefinedType package, its contents will not be included in the list (although the package is loaded and visible in the browser). As with previous versions, you can use select to navigate and select any type defined in the model.

(Default = empty string)

CompareBuildNumberInRepository

The boolean property CompareBuildNumberInRepository can be used to prevent Rhapsody from opening a model that was developed in a different Rhapsody build. This prevents accidental "upgrading" of models.

When this property is set to Cleared, Rational Rhapsody will not open models developed in previous builds, and will display a message to this effect if the user tries to open such a model.

ComponentFileIsSavedUnit

The property ComponentFileIsSavedUnit determines whether or not new component files added to the model are automatically saved as units.

Default = Cleared

ComponentIsSavedUnit

The ComponentIsSavedUnit property determines whether new components are saved as units (separate files) by default.

(Default = Checked)

DefaultDirectoryScheme

The DefaultDirectoryScheme property is used by the hierarchical repository functionality. This property is available only at the project level, but activates or deactivates the Save in Subdirectory check box on the Unit Information for Package window.

The possible values are as follows:

- Flat - All units are stored in the project _rpy directory (as in previous versions of Rational Rhapsody).
- PackageAsDirectory - New packages (and their descendants) are nested in a separate directory, no more than one level below the parent. The package subdirectory has the same name as the package it contains.

(Default = Flat)

DefaultType

The DefaultType property specifies the default data type to be used for any newly created attribute, variable, or argument.

Note: The setting must include the package::usertype for the code to be generated.

You can set the default data type to any type defined by your design and coding standards. For example, you could set the default data type to "package::int32" or to a user-defined type called "not_defined" to emphasize that the type for this attribute, variable, or argument was not yet defined. (Default = empty string)

DefaultUnitFileName

If you merge a unit into a model, and the model already contains a unit with the same filename, Rational Rhapsody automatically adds a numerical suffix to the filename in order to differentiate it from the existing file. This is problematic in scenarios where it is important that the original name of the unit be

maintained.

The property General::Model::DefaultUnitFileName can be used to define a "template" for creating filenames on the basis of element names. In cases where there is a possibility that units is merged from other models, this property can be used to establish unique file-naming schemes before development of the models begins.

For the value of this property, you can combine a model-unique string with the keywords $name, $owner and $pid. If you want the filename to reflect the element it represents, make sure to include the keyword $name in the property value.

For example, you can specify the value of the property as modelA_$name, and this will prevent any naming conflicts if you later bring in elements with the same name from a different model.

If the value of the property is left blank, then Rational Rhapsody uses its default naming scheme in which filenames are taken from the name of the element they represent.

Note that this property only affects filenames for elements that are automatically saved as units as soon as you create them. If you take an element that is not a unit, and select Create Unit from the context menu, the value of this property will not be used.

Default = Blank

DescriptionEditor

The DescriptionEditor property specifies the editor to use when editing element descriptions (for example, Word). See the EditorCommandLine property for information on specifying the editor to use for external code. (Default = empty string)

DescriptionEditorSupportsRTF

The DescriptionEditorSupportsRTF property specifies whether the editor specified in the DescriptionEditor property supports rich text format (RTF). (Default = Cleared)

DescriptionTextLimit

The property DescriptionTextLimit determines the maximum length of the descriptions that can be entered for elements in a Rhapsody model.

The value of the property represents the number of bytes that should be allocated for the description.

Default = 32768

DiagramIsSavedUnit

The DiagramIsSavedUnit property determines whether diagrams are saved as units (separate files) by default.

Default = Cleared

DiagramsToolbar

If you create a custom diagram type, you also have the option of including an icon for the new diagram type in the the Diagrams toolbar.

To add the new type of diagram to the Diagrams toolbar you must modify the value of the DiagramsToolbar property to include the name of the new diagram type in the comma-separated list, for example, OV-1, RpySeparator,RpyDefault (If this property is left empty, the toolbar will inlude only the default icons.)

The strings to use in this list are as follows:

- ActivityDiagram
- CollaborationDiagram
- ComponentDiagram
- DeploymentDiagram
- ObjectModelDiagram
- SequenceDiagram
- Statechart
- StructureDiagram
- UseCaseDiagram
- RpyDefault
- RpySeparator

EditorCommandLine

The EditorCommandLine property enables you to specify which external editor is started when you edit code. If this property is empty, Rational Rhapsody runs the internal editor by default. When using this property, keep in mind the following:

- You can also use the Browse button to locate the text editor.
- The ClassCodeEditor property must be set to CommandLine for this property to take effect.

Alternatively, if you associate your text editor with the file types .h and .cpp in Windows and set the ClassCodeEditor property to Associate, that editor is started when you edit these files. The advantage to this approach is that only one editor session is started. See the DescriptionEditor property for information on specifying the editor to use for element descriptions. (Default = empty string)

EnvironmentVariables

The EnvironmentVariables property enables you to specify environment variables for Rhapsody to execute when your project is opened. You can use environment variables to specify various file paths in your project.

For example, you could use environment variables to specify the location of legacy source files or the location of referenced units. This capability provides the benefit of storing your model and environment in one location (the Rational Rhapsody project) so you can more easily share and distribute complex projects.

You can use environment variables to specify various file paths in your project. For example, you could use environment variables to specify the location of legacy source files or the location of referenced units. This capability provides the benefit of storing your model and environment in one location (the Rational Rhapsody project) so you can more easily share and distribute complex projects.

Rhapsody parses the content of the EnvironmentVariables property and executes the specified environment variables. This execution occurs when a project is opened and after all project-level properties are applied, but before any additional units (packages, components, and diagrams) are read.

When you open a project and all the properties overridden at the project level are applied (but before any additional units are read), Rational Rhapsody parses the contents of this property and sets the relevant environment variables. This setting affects every place Rhapsody looks for an environment variable, which means that it affects the logical path. You can override this property only at the site.prp or project level.

For example, consider the case where you include legacy files in your project, and use the variable LEGACY_DIR in your makefile to specify the location of those files. If you set the EnvironmentVariables property to include the path for LEGACY_DIR, Rational Rhapsody executes the variable when the model is opened so the make utility can expand the LEGACY_DIR variable. In essence, your “environment” is contained in the Rational Rhapsody model.

As another example, consider the case where you have reference units in your model (added using the option Add to Model As Reference). You can edit the location of a reference unit using the Directory field of the Unit Information window, and use an environment variable as part of that location.

If you set the EnvironmentVariables property to include the path of this environment variable, Rational Rhapsody will parse and execute that environment variable when it opens the project, and then search for the reference unit in the specified location. The value of this property is a MultiLine in the following format (with one variable definition per line):

Rhapsody parses the content of the EnvironmentVariables property and executes the specified environment variables. This execution occurs when a project is opened and after all project-level properties are applied, but before any additional units (packages, components, and diagrams) are read.

“Execution” means iterating over the lines and setting the environment variable through the operating system API—there is no “source.”

When you open a project and all the properties overridden at the project level are applied (but before any additional units are read), Rational Rhapsody parses the contents of this property and sets the relevant environment variables. This setting affects every place Rhapsody looks for an environment variable, which means that it affects the logical path.

You can override this property only at the site.prp or project level.

Variable name one=path Variable name two=path ...

For example: COMMON_BASE=C:\SomeDirectory\SomeSubdirectory
ANOTHER_BASE=E:\SomeOtherDirectory\SomeOtherSubdirectory

Note: You can override this property only at the site.prp or project level.

For example, consider the case where you include legacy files in your project, and use the variable LEGACY_DIR in your makefile to specify the location of those files. If you set the EnvironmentVariables property to include the path for LEGACY_DIR, Rational Rhapsody executes the variable when the model is opened so the make utility can expand the LEGACY_DIR variable. In essence, your "environment" is contained in the Rational Rhapsody model.

The following restrictions and limitations apply to this property:

- Comments are not supported.
- There is no way to “unset” the variables, other than exiting Rhapsody.
- Changes in the property setting will take place the next time the project is loaded.
- If you check out a different version of the . rpy file, the environment variables are reset (they are not “unset” first).
- If you read another project where this property is overridden in the project context, that setting will execute on top of the previous settings (the settings are not “unset” first).
- Environment variables defined in “included” property (. prp) files are not supported.
- The value of the EnvironmentVariables property cannot be based on the value of the same property at a higher level. That is, subsequent definitions of the property cannot be based upon the previous definition. Therefore, you cannot define a $BLK_VAR in the site.prp file and then have a new variable $APP_VAR defined in the project (. rpy) file with a value of $BLK_VAR\SubDir, where the $BLK_VAR value is used from the site.prp file EnvironmentVariables definition.
- Rhapsody does not expand environment variables into Rational Rhapsody generated makefiles or build files. Rhapsody does execute the variable when the model is opened and before makefile and build file generation, which enables the make or build utility to assume the responsibility of expanding the environment variable. Most but not all make and build utilities can expand environment variables.

(Default = empty MultiLine)

Extension

The Extension property specifies the extension appended to the project file name. For example, Rational Rhapsody saves the project file for the project Pager as Pager.rpy, and its repository as \Pager_rpy. (Default = rpy)

ExternalImageEditorCommand

The ExternalImageEditorCommand property defines the command line to be used in order to run the user-defined image editor (ex. Microsoft Paint, Paint Shop Pro, etc.).

FileIsSavedUnit

The FileIsSavedUnit property determines whether new files are saved as units (separate files) by default. (Default = Cleared)

Filter

The Filter property contains a list of metaclasses that are filtered from the toolbars. A type that is in this list will not appear on a tool bar.

(Default = empty string)

FolderIsSavedUnit

The property FolderIsSavedUnit determines whether or not new folders added to the model are automatically saved as units.

Default = Cleared

GeneralElementMenuName

The GeneralElementMenuName property is for internal use only. You should not make any changes to this property unless directed by someone from IBM Rational Rhapsody.

Default = General Elements

HelpersFile

The HelpersFile property can be used to associate a .hep file with a model. You can type in the full path of the .hep file or you can use the "..." button to select the .hep file. .hep files are used to store the details of helper applications that have been developed to facilitate working in Rational Rhapsody.

Note that if you specify a .hep file using this property, Rational Rhapsody will not recognize the helper applications defined in the profile-specific .hep file if one is provided for the profile you are using.

(Default = Blank)

HighlightElementsInActiveComponentScope

When this property is Checked, elements within the scope of the active component and configuration are highlighted as bold in the browser.

(Default = Cleared)

ImageEditor

Defines which image editor to use when opening an associated image. Available values are as follows:

- AssociatedApplication - Allows the OS choose according to the extension (Default)
- External - Uses user-defined editor

If the ImageEditor property is set to External, then the property ExternalImageEditorCommand must be defined.

ModelCodeAssociativityFineTune

The ModelCodeAssociativityFineTune property enables you to change the default DMCA mode in the site.prp file. However, you usually set this property using the GUI (by selecting Code > Dynamic model code associativity). The possible values are as follows:

- Bidirectional - Both code generation and round trip are launched automatically for the code view window.
- Roundtrip - Only roundtrip is launched automatically in the online code view windows.
- Code Generation - Only code generation is launched automatically in the online code view windows.
- None - Disables DMCA entirely. The online code view windows become simple text editors.

(Default = Bidirectional)

NamesRegExp

The NamesRegExp property specifies the regular expression describing the format of a legal name of an element. For example, a legal class name might be Class1 but not 1Class. For example, suppose you want to allow spaces, slashes, and dashes in element names.

To do this, add a space after the underscore in the default value, as follows:

^(([a-zA-Z_][a-zA-Z0-9_]*)|(operator.+))$

Note that this change applies to all types of named elements. If you are going to generate code for the element, spaces in some element names are not allowed (for example, class names). Therefore, you would most likely use this property for elements that will never participate in code generation, such as an analysis (not design) package.

(Default = ^(([a-zA-Z_][a-zA-Z0-9_])|(operator.+))$)*

ObjectIsSavedUnit

The ObjectIsSavedUnit property determines whether new packages are saved as units (separate files) by default. (Default = Cleared)

OutputWindowFont

The OutputWindowFont property specifies the font used for messages displayed in the Output window. (Default = Courier New 9 NoBold NoItalic)

PackageIsSavedUnit

The PackageIsSavedUnit property determines whether new packages are saved as units (separate files) by default. (Default = Checked)

PathInProjectList

When a project is added to a project list, the path to the project is added to the project list file (.rpl). The property PathInProjectList can be used to specify whether an absolute or relative path should be used when the project is added to a project list.

The possible values for the property are Absolute and Relative.

Note that when you change the value of this property for a project after the project has already been included in project lists, you have to open the relevant project lists in Rational Rhapsody and select Save All in order to update the project path in the project list files.

Default = Absolute

PredefinedTypesInComboBox

The PredefinedTypesInComboBox property is a comma separated list of the predefined types included in the types combo box. The PredefinedTypesInComboBox property only affects predefined types. Types from other packages are not affected.

(Default = empty string)

RefactorRenameRegExp

This property is used as part of the regular expression string to search for user code instances of the element you are renaming. If you are renaming an attribute, the program needs to find any instances in the user code (such as the operation of a body, action on entry/exit, reaction in a station, overridden properties, and configuration initialization).

The program only inspects the user code because everything else (dependencies and other relations/references) are automatically updated upon renaming the element. However, this feature also performs some additional refactoring. Therefore, the program searches for all instances of the element in user code and shows them in the preview window to the user.

For example, if a user created functions related to a changed attribute and if the new name of the attribute is "attribute_0123," then the program renames those instances in user code to "get_attribute_0123" and "set_attribute_0123" with the "get_" and "set_" prefixes used as they are set by default in this property.

The user may change these default settings, but they need to be compatible with the regular expression syntaxes of both the Rational Rhapsody search and replace, as well as the internal code editor.

(((get_)|(set_)|(its))$keyword)|($keyword)

ReferenceUnitPath

The ReferenceUnitPath property defines how to save a reference unit path.

The property can be set to "Absolute" or "Relative" to specify whether units that are added to the model by reference to use an absolute path or by the relative path.

If the property is set to "Relative," then newly added referenced units contain a path relative to the project directory.

Note:

The correct way to change to a relative path is to set this property and then add the unit to the model again.

If the ReferenceUnitPaths property is set to "Absolute" when the unit is loaded, then Rational Rhapsody continues to expect an "Absolute" path when the model is loaded. Editing the path of the unit to change it from "Absolute" to "Relative" does not work.

RenameUnusedFiles

By default, when you use "Delete from Model" to delete a unit in Rational Rhapsody, the element is removed from the browser but the unit file remains in the project directory. This is also true for actions such as rename and move. The boolean property RenameUnusedFiles allows you to specify that Rational Rhapsody should add an additional file extension to the names of files that remain in the project directory after one of these actions.

To use this feature, set the value of this property to Checked.

Use of this feature makes it easy to identify the unused files in the file system if you would like to delete them at some stage.

By default, the extension added when the property is set to Checked is ".keep". This extension can be changed by modifying the value of the property General::Model::RenameUnusedFilesWith.

Default = Cleared

RenameUnusedFilesWith

If the property General::Model::RenameUnusedFiles is set to True, then Rational Rhapsody adds an additional file extension to the names of files that remain in the file system after actions such as rename and "delete from model" in Rational Rhapsody. The property RenameUnusedFilesWith allows you to specify the extension that you would like Rhapsody to use for this feature.

Default = .keep

ReservedWords

The ReservedWords property is a string that specifies the list of Rational Rhapsody reserved words. Reserved words cannot be used as names of classes, attributes, and so on. To specify additional reserved words for your environment, use the property lang_CG::Environment::AdditionalReservedWords.

The default value is as follows:

asm auto bad_cast bad_typeid break case catch char class const const_cast continue default delete do double dynamic_cast else enum except extern finally float for friend goto if inline int long namespace new operator private protected public register reinterpret_cast return short signed sizeof static static_cast struct switch template this throw try type_info typedef typeid union unsigned using virtual void volatile while xalloc

SAExternalID

This read-only property displays the identity in the encyclopedia of the Rational System Architect imported elements .

SAExternalType

This read-only property displays the type in the encyclopedia of the Rational System Architect imported elements.

SearchPath

The SearchPath property is currently unused. (Default = .)

ShowPotentialUnresolvedReferences

If you delete a model element that is referenced by other elements that are read-only, Rational Rhapsody displays a dialog listing the relevant read-only files so that you can change them to read/write. This is designed to prevent situations where element deletions result in unresolved references.

If you do not want Rhapsody to display this dialog, you can set the value of the property ShowPotentialUnresolvedReferences to Cleared.

Default = Checked

SourceFont

The SourceFont property determines which font is used for source code in the browser and graphic editor windows. You can use only fonts that are actually installed on your system. For example, Courier is a fixed-size font that is available only in certain sizes, but not the 5-point size.

However, Courier New is a TrueType font, which can be used in any size-integer or floating point because it is provided in vector format rather than as a bitmap.

To see which sizes are available on your system, click Choose Font in the specification dialog for the SourceFont property. Available fonts are listed in the resulting Font window. Note that italics and bold are ignored. (Default = Courier New 9 NoBold NoItalic)

Submenu1List

The Submenu1List property controls which elements populate the menu for the Submenu1Name property. By default, Submenu1 concerns diagrams. Therefore, by default, the default values for Submenu1List are the diagrams available in Rational Rhapsody. You can re-order, remove, or re-add any of these elements by doing so through the Submenu1List property.

Note the following:

- Whatever element that is removed from Submenu1List will appear in the middle portion of the Add New menu if that element is relevant for your project. The element must appear somewhere if it is a valid element.
- The Submenu1List and Submenu1Name properties are also used by Tools > Diagrams. When you make a change to Submenu1List, to have it take effect on the Tools menu, you must save your project, close it, and then open it again. In addition, if you delete the Submenu1 value from the SubmenuList property, all the Rational Rhapsody diagram choices will appear in the Tools menu, instead of under Tools > Diagrams (after you save your project and then re-open it again).
- The General::Model::AddNewMenuStructure property overrides this property.

Default = ObjectModelDiagram,SequenceDiagram,UseCaseDiagram,ComponentDiagram,DeploymentDiagram,CollaborationDiagra Diagram,PanelDiagram,IReferencedDiagram,Flowchart

Submenu1Name

The Submenu1Name property controls the name that appears for the Submenu1 group in the Add New menu. Use this property in conjunction with Submenu1List to populate the elements that should appear for the Submenu1 group. The SubmenuList property controls whether Submenu1Name appears on the Add New menu.

Note the following:

- The Submenu1List and Submenu1Name properties are also used by Tools > Diagrams. When you make a change to Submenu1List, to have it take effect on the Tools menu, you must save your project, close it, and then open it again. In addition, if you delete the Submenu1 value from the SubmenuList property, all the Rational Rhapsody diagram choices will appear in the Tools menu, instead of under Tools > Diagrams (after you save your project and then re-open it again).
- The General::Model::AddNewMenuStructure property overrides this property.

Default = Diagrams

Submenu2List

The Submenu2List property controls which elements populate the menu for the Submenu2Name property. By default, Submenu2 concerns relations. Therefore, by default, the default values for Submenu2List are the relations available in Rational Rhapsody. You can re-order, remove, or re-add any of these elements by doing so through the Submenu2List property.

Note the following:

- Whatever element that is removed from a group will appear in the middle portion of the Add New menu if that element is relevant for your project. The element must appear somewhere if it is a valid element.
- The General::Model::AddNewMenuStructure property overrides this property.

Default = Dependency,Derivation,Flow,AssociationEnd,Generalization,Realization,Hyperlink

Submenu2Name

The Submenu2Name property controls the name that appears for the Submenu2 group in the Add New menu. Use this property in conjunction with Submenu2List to populate the elements that should appear for the Submenu2 group. The SubmenuList property controls whether Submenu2Name appears on the Add New menu.

Note: The General::Model::AddNewMenuStructure property overrides this property.

Default = Relations

Submenu3List

The Submenu3List property controls which elements populate the menu for the Submenu3Name property. By default, Submenu3 concerns tables and matrices. Therefore, by default, the default values for Submenu3List are the elements available in Rational Rhapsody for tables and matrices. You can re-order, remove, or re-add any of these elements by doing so through the Submenu3List property.

Note the following:

- Whatever element that is removed from a group will appear in the middle portion of the Add New menu if that element is relevant for your project. The element must appear somewhere if it is a valid element.
- The General::Model::AddNewMenuStructure property overrides this property.

Default = TableLayout,TableView,MatrixLayout,MatrixView

Submenu3Name

The Submenu3Name property controls the name that appears for the Submenu3 group in the Add New menu. Use this property in conjunction with Submenu3List to populate the elements that should appear for the Submenu3 group. The SubmenuList property controls whether Submenu3Name appears on the Add New menu.

Note: The General::Model::AddNewMenuStructure property overrides this property.

Default = Table\Matrix

Submenu4List

The Submenu4List property controls which elements populate the menu for the Submenu4Name property.

By default, Submenu4 concerns annotations. Therefore, by default, the default values for Submenu4List are the elements available for annotations/requirements in Rational Rhapsody. You can re-order, remove, or re-add any of these elements by doing so through the Submenu4List property.

Note the following:

- Whatever element that is removed from a group will appear in the middle portion of the Add New menu if that element is relevant for your project. The element must appear somewhere if it is a valid element.
- The General::Model::AddNewMenuStructure property overrides this property.

Default = Constraint,Component,ControlledFile

Submenu4Name

The Submenu4Name property controls the name that appears for the Submenu4 group in the Add New menu. Use this property in conjunction with Submenu4List to populate the elements that should appear for the Submenu4 group. The SubmenuList property controls whether Submenu4Name appears on the Add New menu.

Note: The General::Model::AddNewMenuStructure property overrides this property.

Default = Annotations

SubmenuList

The SubmenuList property controls which submenu groups appear at the bottom portion of the Add New menu. Use this property in conjunction with its corresponding Submenu#Name and Submenu#List properties (for example, Submenu1Name and Submenu1List).

For example, if you remove the "Submenu1," value, which is related to diagrams (see the Submenu1Name and Submenu1List properties); then the "Diagrams" group will not appear on the Add New menu.

Note the following:

- Removing the "Submenu1" value has an effect on Tools > Diagrams. After doing so, after you save your project and then open it again, all the Rational Rhapsody diagram choices will appear in the Tools menu, instead of under Tools > Diagrams (which is what happens when the "Submenu1" value is set in the SubmenuList property).
- The General::Model::AddNewMenuStructure property overrides this property.

Default = Submenu1,Submenu2,Submenu3,Submenu4

TcSE_LOID

This property is used only if you have the Teamcenter Systems Engineering (TcSE) add-on installed. If you have TcSE installed, Rational Rhapsody stores the ID of its corresponding TcSE element in this property.

TypeComboBoxSort

The TypeComboBoxSort property determines how the Type ComboBox is sorted. Listed values can be sorted one of two ways: Alphabetically or ByPackage. If the listed values are sorted ByPackage, the contents of the drop-down list are sorted by their respective packages and within each package hierarchy, the types are sorted alphabetically.

UndoBufferSize

The UndoBufferSize property is an integer that specifies how many undo transactions are remembered by Rhapsody. A value of "0" means no undo transactions are remembered, "1" means one undo transaction is remembered, and so on. (Default = 20)

UnresolvedSymbol

By default, Rational Rhapsody displays "(U)" next to unresolved model elements in the browser and in diagrams.

The property UnresolvedSymbol allows you to specify a different symbol to use to indicate unresolved model elements. Just enter the string that you would like Rhapsody to use.

Default = (U)

UseIncrementalSave

By default, the Rational Rhapsody Save option saves only the units that have been modified. The property UseIncrementalSave allows you to specify that the Save option should save the entire model.

To have Rhapsody save the entire model, set this property to Cleared.

Default = Checked

WarnForDuplicates

The WarnForDuplicates property specifies whether Rational Rhapsody should issue a warning message when you add an element to the model with a name that is identical to an already existing name for the same kind of element.

Within a given project, you can have two packages with the same name, or two classes or objects with the same name (for example, P1::p and P2::p), provided they are in different scopes.

(Default = Checked)

ModelLibraries

The metaclass ModelLibraries contains properties related to reference models such as the Java reference model.

JavaAPIPackage

The property JavaAPIPackage is used to specify the location of the Java reference model that is included with Rational Rhapsody. This is a model of the classes contained in Java SE 6. Rhapsody uses the property when you select the "Add Java API Library" item from the File menu.

Default = $OMROOT/LangJava/JDKRefModel/JDKModel_rpy/java.sbs

Profile

The Profile metaclass contains a property that specifies the behavior of profiles.

AutoCopied

The AutoCopied property specifies a comma-separated list of physical paths of profiles that are automatically copied into a newly created project (using Add To Model with As Unit). (Default = empty string)

AutoReferences

The AutoReferences property specifies a comma-separated list of physical paths to profiles that will automatically be referenced by new projects when they are created (using the Add to Model, As Reference functionality). (Default = empty string)

Relations

The Relations metaclass contains a property that controls the default multiplicity of relations.

DefaultMultiplicity

The DefaultMultiplicity property specifies the default multiplicity for relations for which the multiplicity is not specified. (Default = 1)

Report

The Report metaclass controls the attributes of the Rational Rhapsody internal reporter.

ExternalViewerCommand

The ExternalViewerCommand property supplies the command to use to run an external RTF viewer. The command must take the format: "<<executable name>>" "$fileName"

This property is only used if the ReportViewer property is set to "External."

ReportViewer

The ReportViewer property specifies which RTF viewer to use in order to show the generated "Report on model" RTF document. Available options are as follows:

- Rhapsody - Uses the Rational Rhapsody internal view
- Associated - Allows the OS to choose the right viewer according to the file extension
- External - Uses the command line set in the ExternalViewerCommand to run an external viewer program

(Default = Rhapsody)

RTFCharacterSet

The RTFCharacterSet property enables you to define the necessary character set used by the RTF format file created by the Rational Rhapsody internal reporter.

The character set is used in the RTF multilanguage and description styles, which are used for the Name Label and Description fields of the report. The RTF file created by the Rational Rhapsody internal reporter should include a specific character set for each language.

For example, set this property to “ \fcharset128” for Japanese.

The default value, an empty string, preserves the current behavior. You can define this property on the project and higher (site or factory) level. (Default = empty string)

ReporterPLUS

The ReporterPLUS metaclass controls the behavior of ReporterPLUS.

ReportAll

The ReportAll property is used by the ReporterPLUS tool. Do not change the value of this property.

The default value is as follows:

"$OMROOT/../Reporter/Reporter.exe" /m "$modelname" /l "reg")

ReportSelected

The ReportSelected property is used by the ReporterPLUS tool. Do not change the value of this property.

The default value is as follows:

"$OMROOT/../Reporter/Reporter.exe" /m "$modelname" /s "$scope" /l "reg"

SplitDiagrams

The SplitDiagrams property specifies whether to split large reports across pages when they are exported (as metafiles) to other tools, such as the Rational Rhapsody internal reporter or the API.

By splitting large diagrams across multiple pages, you improve their readability.

This feature does not apply to the following operations:

- Printing diagrams
- Copying diagrams into the clipboard
- Pasting them from the clipboard

It applies only to exporting diagrams (as metafiles) to other tools. The possible values are as follows:

- True - Divide the diagram vertically or horizontally as needed when exporting. The diagram is split if fitting the diagram onto a single page will require a zoom factor of 65% or lower.
- False - Zoom out as necessary to fit the diagram on a single page. This is the default behavior for Rhapsody Version 4.0 and its point releases.

If this property is set to True, the diagram is first split by column (top to bottom), then by row. The following figure shows the order in which the pages are created.

For example, the following figure shows a sequence diagram forced onto a single page. The following figures show how this sequence diagram would be split into multiple pages. The following sections describe implementation-specific behavior.

When a sequence diagram is split into multiple pages, the names of instances (the upper pane of the SD) is added to the top of each page. Rational DOORS If you have opened a diagram and want to see all of the pages, select Edit OLE Object Document Object Open.

API In previous versions of Rational Rhapsody, you exported a diagram in one metafile using the

following call:

HRESULT getPicture ([in] BSTR fileName); Version 4.1 introduces a new method, getPictureAsDividedMetafiles. The syntax is as follows: HRESULT getPictureAsDividedMetafiles ([in] BSTR firstFileName, [out, retval] IRPCollection** fileNames); In the call, firstFileName specifies the naming convention for the created files.

For example, if you passed the value “Foo” as the firstFileName:

- If the diagram can be drawn on one page, the name of the metafile is Foo.
- If the diagram is split into multiple pages, the first file is named FooZ_X_Y. The variables used in the name are as follows:
- Z - The number of the created file
- X - The number of the page along the X vector
- Y - The number of the page along the Y vector
- For example, the file Foo2_1_2 means that this is the second metafile created and it contains one page, which is the second page along the Y vector (the X vector is 1).

All the file names is inserted in the sent strings list (fileNames). (Default = Checked)

TemplateEditor

The TemplateEditor property is used by the ReporterPLUS tool. Do not change the value of this property.

The default value is as follows:

"$OMROOT/../Reporter/Reporter.exe" /l "pro"

Workspace

The Workspace metaclass contains properties that control the behavior of the Rational Rhapsody workspace.

AnimationOutputBufSize

The AnimationOutputBufSize property specifies the size, in bytes, of the output buffer used by animation. (Default = 65536 bytes)

ApplyHiddenSubjects

This property allows the user to hide subjects specified in the "HiddenSubjects" property. However, when the user is running the Rational Rhapsody Modeler or Rhapsody Corproate, this property is ignored.

(Default = Cleared)

DoubleClickOnRelationsShould

The DoubleClickOnRelationShould specifies what action to perform when double-clicking on an item in the Relations window. Possible values are as follows:

- OpenFeatures - Open the Features Dialog of the item.
- LocatetheElement - Highlight the item in the browser. If the item is a diagram, the diagram is opened.
- DoBoth - Open the Features Dialog Box and highlight the item in the browser.

GenerateNameFromLabelInLabelMode

This property indicates whether the element name should be generated from its label when working in the "Label" mode. Label mode can be set by selecting the View > Label Mode menu option.

(Default = Checked)

HiddenSubjects

This is a comma separated list of subjects that need to be hidden from display in the features window. The specified subjects are only hidden if the "ApplyHiddenSubjects" property is set to Checked.

However, when the user is running Rhapsody Modeler or Rhapsody Corporate, the value of the "ApplyHiddenSubjects" property is ignored and by default all of the subjects specified in this list are hidden in the Features window.

OkMayDockFeatures

The OkMayDocFeatures property specifies whether the features window is set to Show Mode (Checked). In Show Mode, if the features window is docked and you double-click an element (in either the browser or drawing area), the window will float. Pressing OK will dock it again instead of closing it.

(Default = Cleared)

OpenDiagramWithLastPlacement

The OpenDiagramWithLastPlacement property is a Boolean value that determines whether Rational Rhapsody will display your diagrams using the last values of the following diagram properties:

- Size
- Position (relative to the upper, left-hand corner)
- Status (maximized, minimized, and so on)
- Zoom factor
- Scroll location

(Default = Checked)

OpenWindowsWhenLoadingProject

The OpenWindowsWhenLoadingProject project is a Boolean value that determines whether Rational Rhapsody should load the window configuration information saved from a previous session.

Rhapsody saves a user's window configuration for a given project in the workspace file (.rpw) each time a project is closed. The information saved includes window size, position, status of feature windows, and the scaling or zoom factor of open diagrams.

To prevent Rhapsody from loading that information the next time the project is opened, set this property to Cleared.

This property used to be called General::Workspace::OpenWorkspaceWhenLoadingProject. It was changed because workspaces now store information on loaded units (for the partial load feature), as well as window preferences. This property affects only the windows.

(Default = Checked)

OpenWorkspaceWhenLoadingProject

The OpenWorkspaceWhenLoadingProject project (under General::Workspace in Rational Rhapsody Developer for C and J) is a Boolean value that determines whether Rational Rhapsody should automatically load the workspace when it loads the project.

(Default = Checked)

ShowLabelInFeaturesDialog

The property indicates whether the Features dialog should display the Label instead of the Name. (Default = Cleared)

IntelliVisor

The IntelliVisor subject controls the IntelliVisor tool, which provides suggestions during common tasks. It includes the following metaclasses:

- General - Contains properties that specify when the IntelliVisor is enabled.
- PredefineMacros - Contains properties that specify the syntax of the predefined macros included in the lists generated by the IntelliVisor.
- PredefineMacrosTooltip - Contains properties that specify the tooltips displayed for the predefined macros.

General

The General metaclass contains properties that determines when the IntelliVisor is enabled in Rational Rhapsody.

ActivateOnCode

The ActivateOnCode property determines whether the Intellivisor is enabled (Checked) or disabled (Cleared) for code.

Default = Checked

ActivateOnGe

The ActivateOnGe property determines whether the Intellivisor is enabled (Checked) or disabled (Cleared) in the drawing area.

Default = Checked

ShowPredefineMacros

The ShowPredefineMacros property determines whether the values defined in the IntelliVisor::PredefineMacros properties (and their corresponding tooltips defined in IntelliVisor::PredefineMacrosTooltip) are included in the lists generated by the IntelliVisor. The default items defined in PredefineMacros and PredefineMacrosTooltip (GEN, IS_IN, and OPORT) are commonly used values. However, you can expand this list by doing the following:

- Add a property to the predefined macros list. The name of the property is visible in the list control; the value of the property is placed inside the code when the macro is selected.
- Add a tooltip to the tooltip list. This tooltip is visible when the item is selected from the list.

Default = Checked

PredefineMacros

The PredefineMacros metaclass contains properties that specify the syntax of the predefined macros that are included in the lists generated by the IntelliVisor.

CGEN

The CGEN property specifies the syntax of the macro that generates an event.

Default = CGEN

CIS_IN

The CIS_IN property specifies the syntax of the macro that determines whether a statechart is in the specified state.

Default = CIS_IN

GEN

The GEN property specifies the syntax of the macro that generates an event.

Default = GEN

IS_IN

The IS_IN property specifies the syntax of the macro that determines whether a statechart is in the specified state.

Default = IS_IN

OPORT

The OPORT property is a shortcut for OUT_PORT. This macro relays messages through the port. For example: OPORT(p)-foo(); // calls foo() via the port // OPORT(p)-GEN(evt); // sends event evt via the port

Default = OPORT

PredefineMacrosTooltip

The PredefineMacrosTooltip metaclass contains properties that specify the tooltips displayed for the predefined macros that are included in the lists generated by the IntelliVisor.

CGEN

The CGEN property specifies the tooltip displayed by the IntelliVisor for the CGEN macro.

Default = CGEN(<<instance>>,<<event>>)

CIS_IN

The CIS_IN property specifies the tooltip displayed by the IntelliVisor for the CIS_IN macro.

Default = CIS_IN(<<me>>,<<state>>)

GEN

The GEN property specifies the tooltip displayed by the IntelliVisor for the GEN macro.

Default = Event generation macro:GEN(<<event>>)

IS_IN

The IS_IN property specifies the tooltip displayed by the IntelliVisor for the IS_IN macro.

Default = Statechart test macro:IS_IN(<<state>>)

OPORT

The OPORT property specifies the tooltip displayed by the IntelliVisor for the OPORT macro.

Default = Port macro:OPORT(<<p>>)

Java(1.1)Containers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMContainers subject contain the following metaclasses:

- BoundedOrdered - Defines the properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines the properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- EmbeddedFixed - Defines the properties for implementing embedded fixed relations.
- EmbeddedScalar - Defines the properties for implementing embedded scalar (one-to-one) relations.
- Fixed - Defines the properties for implementing relations of fixed size.
- General - Defines the properties that set the directives and include files for the container.
- Qualified - Defines the properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines the properties for implementing scalar relations.
- StaticArray - Defines the properties for implementing static arrays.
- UnboundedOrdered - Defines the properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines the properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines the properties for user-defined implementations of relations.
- You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.
- For example, you can change the definition of the Implementation property as follows: Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname.addElement($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = new $CType().

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType().

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is Vector.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is ($RelationTargetType)($cname.elementAt($index)).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

(Default = strong)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

The default is $Create.

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

(Default = $IterType $iterator = 0;)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is ($RelationTargetType)($cname.elementAt($iterator)).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $cname.size().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property

body.

The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname.removeElement($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

The default is $cname.removeAllElements().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation,

passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname.addElement($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = new $CType().

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType().

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is Vector.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is ($RelationTargetType)($cname.elementAt($index)).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and

the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

The default is $Create.

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator = 0;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is ($RelationTargetType)($cname.elementAt($iterator)).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $cname.size().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(),

$cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname.removeElement($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

The default is $cname.removeAllElements().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Fixed

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname.addElement($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = new $CType().

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is Vector.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is ($RelationTargetType)($cname.elementAt($index)).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:

$cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL The default is $Create.

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator = 0;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is ($RelationTargetType)($cname.elementAt($iterator)).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $cname.size().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname.removeElement($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

The default is $cname.removeAllElements().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

General

The General metaclass contains properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

(Default = java.util.)*

Qualified

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname.put($keyName,$item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = new $CType().

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType().

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is Hashtable.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

The default is ($RelationTargetType)($cname.get($keyName)).

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The

property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL The default is $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $IterReset.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is ($RelationTargetType)($iterator.nextElement()).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

The default is $iterator.nextElement().

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterCreate.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterCreate.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = $cname.elements().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator.hasMoreElements().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is Enumeration.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other

subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows:

$IterCreate; while (iter.hasMoreElements()) { Object key = iter.nextElement(); if ($cname.get(key).equals($item)) { $cname.remove(key); break; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container.

The default is $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

The default is $cname.remove($keyName).

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Scalar

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $RelationTargetType.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $RelationTargetType.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item (Default)

StaticArray

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is as follows:

$Loop { if($cname[pos] == null) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $target[$multiplicity].

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $RelationTargetType[] $cname.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is $cname[$index].

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL The default is as follows:

$Create; $Loop { $cname[pos] = null; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator = 0;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is $cname[$iterator].

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For

example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $multiplicity.

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for Java is as follows: for (int pos = 0; pos < $multiplicity; pos++)

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos) The default is as follows:

$Loop { if($cname[pos] == $item) { $cname[pos] = null; break; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname-clear()

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

The default is $cname[$index] = $item.

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname.addElement($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = new $CType().

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType().

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is Vector.

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is ($RelationTargetType)($cname.elementAt($index)).

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The

property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL The default is $Create.

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $IterReset;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is ($RelationTargetType)($cname.elementAt($iterator)).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $cname.size().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname.removeElement($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear() The default is $cname.removeAllElements().

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname.addElement($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target* The default is $cname = new $CType().

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname() The default is $Create.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType().

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is Vector.

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index) The default is ($RelationTargetType)($cname.elementAt($index)).

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL The default is $Create.

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

The default is $CreateStatic.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item

in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin() The default is $IterType $IterReset;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is ($RelationTargetType)($cname.elementAt($iterator)).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin() The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $cname.size().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos) The default is $cname.removeElement($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname-clear() The default is $cname.removeAllElements().

User

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container’s at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for “finding” where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Java(1.2)Containers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The Java(1.2)Containers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Defines the properties that set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows:

Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = ArrayList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)($cname.get($iterator))

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $cname.size()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $IterType _pos = $cname.indexOf($item); if (_pos != -1) { $cname.remove(_pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = ArrayList

Find

The Find property specifies the command used to locate an item in a container. For example, the following

command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)($cname.get($iterator))

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various

containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $cname.size()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $IterType _pos = $cname.indexOf($item); if (_pos != -1) { $cname.remove(_pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add(0, $item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = ArrayList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)($cname.get($iterator))

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $cname.size()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $IterType _pos = $cname.indexOf($item); if (_pos != -1) { $cname.remove(_pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty string

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

*Default = java.util.**

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.put($keyName,$item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = HashMap

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default = ($RelationTargetType)($cname.get($keyName))

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)($cname.get($iterator.next()))

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = $iterator.next()

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterCreate

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterCreate

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.keySet().iterator()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = Iterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $IterCreate; while(iter.hasNext()) { Object key = iter.next(); if ($cname.get(key).equals($item)) { $cname.remove(key); break; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = $cname.remove($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $RelationTargetType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default = $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $Loop { if($cname[pos] == null) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = $CreateStatic

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $target[$multiplicity]

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType[] $cname

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create; $Loop { $cname[pos] = null; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate;

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname[$iterator]

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $multiplicity

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos < $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $Loop { if($cname[pos] == $item) { $cname[pos] = null; break; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty MultiLine

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed:

$cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

Default = $cname[$index] = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = LinkedList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding"

where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default = $cname.listIterator($cname.lastIndexOf($cname.getLast()))

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following

command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set

OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)($iterator.next())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = LinkedList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname.indexOf($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $cname = new $CType()

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)($iterator.next())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With

OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = Empty string

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection:
$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default = Empty string

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = Empty string

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the

container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Java(1.5)Containers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The Java(1.5)Containers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Defines the properties that set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows:

Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = ArrayList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname.get($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = $cname.listIterator()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname.get($iterator.nextIndex())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator.next()

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed:

$cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = ArrayList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname.get($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = $cname.listIterator()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname.get($iterator.nextIndex())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = $iterator.next()

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

$cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add(0, $item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = ArrayList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default =

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname.get($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = $cname.listIterator()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname.get($iterator.nextIndex())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator.next()

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling

code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty string

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

*Default = java.util.**

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.put($keyName,$item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = HashMap<$keyType, $RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = ($RelationTargetType)($cname.get($index))

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default = ($RelationTargetType)($cname.get($keyName))

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname.get($iterator.next())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterCreate

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterCreate

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.keySet().iterator()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With

OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = Iterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $IterCreate; while(iter.hasNext()) { Object key = iter.next(); if ($cname.get(key).equals($item)) { $cname.remove(key); break; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = $cname.remove($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $RelationTargetType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default = $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $Loop { if($cname[pos] == null) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = $CreateStatic

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $target[$multiplicity]

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType[] $cname

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create; $Loop { $cname[pos] = null; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate;

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname[$iterator]

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $multiplicity

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos < $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $Loop { if($cname[pos] == $item) { $cname[pos] = null; break; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty MultiLine

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed:

$cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

Default = $cname[$index] = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = LinkedList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname.get($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding"

where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default = $cname.listIterator($cname.lastIndexOf($cname.getLast()))

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following

command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Create

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = $cname.listIterator()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set

OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname.get($iterator.nextIndex())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator.next()

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname.add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = new $CType()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $Create

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType()

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = LinkedList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname.indexOf($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname.get($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $cname = new $CType()

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = $CreateStatic

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = $cname.listIterator()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname.get($iterator.nextIndex())

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator.next()

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = $cname.listIterator(0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With

OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator.hasNext()

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = ListIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname.remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname.clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = Empty string

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container end() operation to locate the last item in the collection:
$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default = Empty string

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default = Empty string

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the

container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

JAVA_CG

The JAVA_CG subject contains several metaclasses for operating system environments and the following general metaclasses:

- Component
- AnimInstrumentation
- Attribute
- Class
- Configuration
- Dependency
- File
- Framework
- JDK
- Operation
- Package
- Port
- Relation
- Type

AnimInstrumentation

The AnimInstrumentation metaclass contains a property that controls the headers for Java files.

Headers

The Headers property is a string that enables you to specify additional #import statements needed for the instrumented code.

Default =
com.ibm.rational.rhapsody.animation.,com.ibm.rational.rhapsody.animcom.*,com.ibm.rational.rhapsody.animcom.animMe*

Argument

The Argument metaclass contains properties that control how arguments are generated in code.

ClassWide

The ClassWide property determines whether a class-wide modifier is generated for the argument. (Default = False)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, $Arguments - The operation argument’s description triggered operations, $Signature - The operation signature constructors, and destructors Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- $Tag - The value of the specified element’s tag
- $Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

(Default = empty string)

Attribute

The Attribute metaclass contains properties that control attributes of code generation, such as whether to generate accessor operations.

Accessor

The Accessor property is ignored by Rhapsody.

AccessorGenerate

The AccessorGenerate property specifies whether to generate accessor operations for attributes. The possible values are as follows:

Checked - A get() method is generated for the attribute. (Default)

Cleared - A get() method is not generated for the attribute.

Setting this property to Cleared is one way to optimize your code for size.

AccessorVisibility

The AccessorVisibility property specifies the access level of the generated accessor for attributes. This enables you to define the access level of an accessor operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute's access level for the accessor.
- public - Set the accessor access level to public.
- private - Set the accessor access level to private.
- default - Set the accessor access level to default.

Default = fromAttribute

AttributeInitializationFile

The AttributeInitializationFile property specifies how static const attributes are initialized. In Rhapsody, you can initialize these attributes in the specification file or directly in the initialization file. This property is analogous to the VariableInitializationFile property for global const variables. The possible values are as follows:

- Default - The attribute is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize constant attributes in the implementation file.
- Specification - Initialize constant attributes in the specification file.

(Default = Default)

ConstantVariableAsDefine

This property is a Boolean value that determines whether the variable, defined as constant in file or package, is generated using a #define macro. Otherwise, it is generated using the const qualifier.

(Default = Cleared)

DeclarationPosition

The DeclarationPosition property enables you to control the declaration order of attributes. The possible values are as follows:

- Default - Similar to the AfterClassRecord setting, with the following difference:
- For static attributes defined in a class with the property Ada_CG::Attribute::Visibility set to Public, these attributes are generated after types whose Ada_CG::Type::Visibility property is set to Public.
- You should not use this setting for new models. See the Rational Rhapsody Developer for Ada documentation for more information.
- BeforeClassRecord - Generate the attribute immediately before the class record.
- AfterClassRecord - Generate the attribute immediately after the class record.
- StartOfDeclaration - Generate the attribute immediately after the start of the section (private or public part of the specification, or package body).
- EndOfDeclaration - Generate the attribute immediately before the end of the section (private or public part of the specification, or package body).

(Default = Default)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

(Default = empty string)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef_DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Default = Empty MultiLine

InitializationStyle

The InitializationStyle property specifies the initialization style used for attributes. When you specify an initial value for an attribute, Rational Rhapsody initializes the attribute based on the value of this property.

In Rational Rhapsody Developer for Java, the possible values are as follows:

- InClass - Initialize the attribute in the class declaration. (Default)
- InConstructor - Initialize the attribute in each of the class constructors.

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for Java Because inlining has no meaning in Java, the Inline property is set to none. (Default = none)

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased. (Default = False)

IsMutable

The boolean property IsMutable allows you to specify that an attribute is a mutable attribute. (Default = False)

IsTransient

The property IsTransient allows you to specify that an attribute should be declared as transient in order to prevent it from being serialized.

Default = Cleared

IsVolatile

The property IsVolatile allows you to specify that an attribute should be declared as volatile.

Default = Cleared

JavaAnnotation

The property JavaAnnotation is used by the Rational Rhapsody code generator to insert Java annotations into generated code.

This property is used primarily for regenerating code that was reverse engineered. When you reverse

engineer code that contains Java annotations, the value of the property JAVA_ReverseEngineering::ImplementationTrait::ImportJavaAnnotation determines how Rhapsody handles the annotation code. If the value of this property is set to Verbatim, then Rational Rhapsody does not import annotations as model elements. Rather, the annotation code is stored as the value of the property JavaAnnotation. When code is later regenerated, it will include the code that was stored in this property.

Default = Blank

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual.
- abstract - Class operations and accessor/mutator are pure virtual.

Default = common

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

Mutator

The Mutator property is ignored by Rhapsody.

MutatorGenerate

The MutatorGenerate property specifies whether to generate mutators for attributes. The possible values are as follows:

- Smart - Mutators are not generated for attributes that have the Constant modifier.
- Always - Mutators are generated, regardless of the modifier.
- Never - Mutators are not generated.

Default = Cleared

MutatorVisibility

The MutatorVisibility property specifies the access level of the generated mutator for attributes. This enables you to define the access level of a mutator operation regardless of the visibility of the attribute. The possible values are as follows:

- fromAttribute - Use the attribute access level for the mutator.
- public - Set the mutator access level to public.
- private - Set the mutator access level to private.
- protected - Set the mutator access level to protected.
- default - Set the mutator access level to default.

Default = fromAttribute

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code. See the Rational Rhapsody Help for detailed information about using composite types. (Default = "")*

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you could add the @Deprecated annotation for an element by entering @Deprecated and a new line as the value of this property.

Default = Blank

VariableInitializationFile

The VariableInitializationFile property specifies how global constant variables are initialized. You can initialize these variables in the specification file. You can use these variables as compile-time constants that can be used to define array sizes, for example. Rhapsody automatically identifies constant variables with const. By modifying this property, you can choose the initialization file directly. The possible values are as follows:

- Default - The variable is initialized in the specification file if the type declaration begins with const. Otherwise, the variable is initialized in the implementation file.
- Implementation - Initialize global constant variables in the implementation file.
- Specification - Initialize global constant variables in the specification file.

(Default = Default)

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The Visibility setting has the following applicability:

- Classes - Applies only to nested classes, which are defined inside other classes.
- Types - Applies only to types that are defined inside classes. It does not apply to global types, which are defined in packages.

The following table lists the visibility for the JAVA_CG subject.

- Protected - The model element is protected.
- Private - The element is private.

Default = fromAttribute

Class

The Class metaclass contains properties that affect the generated classes.

AccessTypeName

The AccessTypeName property specifies the name of the access type generated for the class record. (Default = empty string)

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active classes. The possible values are as follows:

- A string - Specifies the message queue size for an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

Default = Empty string

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects. The possible values are as follows:

- Any integer - Specifies that a stack of that size is allocated for active objects.
- An empty string (blank) - If not specified, the stack size is set in an operating system-specific manner, based on the value of the ActiveStackSize property for the framework.

Default = Empty string

ActiveThreadName

The ActiveThreadName property specifies the name of the active thread. This facilitates debugging in complex environments in which many threads are constantly being created and deleted on-the-fly. This property is effective only in the pSoSystem (both PPC and X86) and VxWorks environments. In pSoSystem, the thread name is truncated to three characters. The animation thread name is not taken from the active thread name. The possible values are as follows:

- A string - Names the active thread.
- An empty string (blank) - The value is set in an operating system-specific manner, based on the value of the ActiveThreadName property for the framework.

Default = Empty string

ActiveThreadPriority

The ActiveThreadPriority property specifies the priority of active class threads. The possible values are as follows:

- A string - Specifies thread priority of an active class.
- An empty string (blank) - The value is set in an operating system-specific manner.

Default = Empty string

AdditionalBaseClasses

The AdditionalBaseClasses property enables you to add inheritance from external classes to the model.

Default = Empty string

Java Specifics In Rational Rhapsody Developer for Java, an inheritance relation is assumed to mean implementing an interface rather than extending it. For example, if you set the AdditionalBaseClasses property to javax.swing.Jtree, the resulting code would be: public class MyClass implements javax.swing.Jtree In other words, MyClass is treated like an interface. In order to extend rather than implement the base class, you must add the string “extends” to the property. For example, if you set AdditionalBaseClasses to extends javax.swing.Jtree, the resulting code would be: public class MyClass extends javax.swing.JTree A third option would be to enter something like: extends javax.swing.Jtree implements Runnable In this case, the resulting code would be: public class MyClass extends javax.swing.Jtree implements Runnable

AdditionalNumberOfInstances

The AdditionalNumberOfInstances property is a string that specifies the size of the local heap allocated for events when the current pool is full. Triggered operations use the event properties. This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during runtime. All events are dynamically allocated during initialization. Once allocated, a thread’s event queue remains static in size. The possible values are as follows:

- An empty string (blank) - No additional memory is to be allocated when the initial memory pool is exhausted.
- n (a positive integer) - Specifies the size of the array allocated for additional instances.

Default = Empty string

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.

- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

BaseNumberOfInstances

The BaseNumberOfInstances property is a string that specifies the size of the local heap memory pool allocated for either:

- Instances of the class (CPP_CG::Class)
- Instances of the event (CPP_CG::Event)
- This property provides support for static architectures found in hard real-time and safety-critical systems without memory management capabilities during run time. All instances of events are dynamically allocated during initialization. Once allocated, a thread's event queue remains static in size.

Triggered operations use the properties defined for events. When the memory pool is exhausted, an additional amount, specified by the AdditionalNumberOfInstances property, is allocated. Memory pools for classes can be used only with the Flat statechart implementation scheme. The possible values are as follows:

- An empty string (blank) - Memory is always dynamically allocated.
- n (positive integer) - An array is allocated in this size for instances.

The related properties are as follows:

- AdditionalNumberOfInstances - Specifies the number of instances to allocate if the pool runs out.
- ProtectStaticMemoryPool - Specifies whether the pool should be protected (to support a multithreaded environment)
- EmptyMemoryPoolCallback - Specifies a user callback function to be called when the pool is empty. This property should be used instead of the AdditionalNumberOfInstance property for error handling.
- EmptyMemoryPoolMessage - When set to true, this property causes a message to be displayed if the pool runs out of memory in instrumented mode.

Default = Empty string

ComplexityForInlining

The ComplexityForInlining property specifies the upper bound for the number of lines in user code that are allowed to be inlined. User code is the action part of transitions in statecharts. For example, using the value 3, all transitions with actions consisting of three lines or fewer of code are automatically inlined in the calling function. Inlining is replacing a function call in the generated code with the actual code statements that make up the body of the function. This optimizes the code execution at the expense of an increase in code size. For example, increasing the number of lines that can be inlined from 3 to 5 has shortened the code execution time in some cases up to 10%. This property applies only to the Flat implementation scheme for statecharts.

Default = 0

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to “MYDLL_API.” The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class.

Default = Empty string

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

(Default = empty string)

Destructor

The Destructor property controls the generation of virtual destructors in C++. The property exists for C for historical reasons, with a single value of auto, but it has no effect on the generated C code. The possible values are as follows:

- auto - A virtual destructor is generated for an object only if it has at least one virtual function.
- virtual - A virtual destructor is generated in all cases.
- abstract - A virtual destructor is generated as a pure virtual function.
- common - A nonvirtual destructor is generated.

(Default = auto)

Embeddable

The Embeddable property is a Boolean property that specifies whether a class can be allocated by value (nested) inside another class or package. For example, if the Embeddable property is True, 20 instances of a class A can be allocated inside another class using the following syntax: A itsA[20]; The possible values are as follows:

- True - The object can be allocated by value inside a composite object or package. The object declaration and definition are generated in the specification file of the composite.
- False - The object cannot be embedded inside another object. The object declaration and definition are generated in the implementation file of the composite.

The Embeddable property is used with the EmbeddedScalar and EmbeddedFixed properties to determine how to generate code for an embedded object. The Embeddable property must be set to True for either of those properties to take effect. It is also closely related to the ImplementWithStaticArray property, which also needs to be set in order to support by-value allocation. To generate C-like code in C++, set the Embeddable property to True. Relations can be generated by value only under the following circumstances:

- The Embeddable property of the nested class is set to True.
- The multiplicity of the relation is well-defined (not “*”).
- The ImplementWithStaticArray property of the component relation is set to FixedAndBounded.

When the Embeddable property is False:

- The attributes of the object are encapsulated. Clients of the object are forced to use it only via its operations, because there is no direct access to its attributes.
- Dynamic allocation must be used. The compiler does not know how to statically allocate an object when its declaration is not visible.
- The nested object cannot be reactive. This is because of the reactive macros. There is a complex workaround for this issue.

(Default = Checked)

EnableDynamicAllocation

The EnableDynamicAllocation property specifies whether to use dynamic memory allocation for objects. The possible values are as follows:

- True - Dynamic allocation of events is enabled. Create() and Destroy() operations are generated for the object or object type.
- False - Events are dynamically allocated during initialization, but not during run time. Create() and Destroy() operations are not generated for the object. This setting is recommended for static architectures that do not use dynamic memory management during run time.

If you are managing your own memory pools, set this property to False and call CPPReactive_gen() directly. The following example shows how to call RiCReactive_gen() directly to send a static event to a reactive object A, when using a member function of A genStaticEv2A(): void A_genStaticEv2A(struct A_t const me) { { /*#[operation genStaticEv2A() */ static struct ev _ev; ev_Init(_ev); RiCEvent_setDeleteAfterConsume(((RiCEvent*)_ev), RiCFALSE); (void) RiCReactive_gen(me-ric_reactive, ((RiCEvent*)_ev), RiCFALSE); /*#]*/ } } Alternatively, you can use internal memory pools by setting the property BaseNumberOfInstances, which results in the use of framework memory pools. If you use the framework memory pools, do not disable the Create() and Destroy() methods because these methods are used to manage the memory pool. When you disable the generation of the Create() and Destroy() methods, you can still inject events in animation by supplying an alternate function to get an event instance. To do this, set the AnimInstanceCreate property. (Default = Checked)*

EnableUseFromCPP

The EnableUseFromCPP property specifies whether to wrap C operations with an appropriate extern C {} wrapper to prevent problems when code is compiled with a C++ compiler. Wrapping C code with extern C enables you to include C code in a C++ application. Note that the structure definition for the object is not wrapped - only the functions are. For example, if the EnableUseFromCPP is set to True for an object, the following wrapper code is generated for its operations:

#ifdef __cplusplus extern "C" { #endif /* __cplusplus */ /* Operations */ #ifdef __cplusplus } #endif /* __cplusplus */

(Default = False)

Final

The Final property, when set to False, specifies that the generated record for the class is a tagged record. This property applies to Ada95. (Default = False)

GenerateAccessType

The GenerateAccessType property determines which access types are generated for the class. The possible values are as follows:

- None - Access types are not generated.
- Standard - An access type is generated.
- General - General access types are generated.

(Default = General)

GenerateDestructor

The GenerateDestructor property specifies whether to generate a destructor for a class.

Default = Cleared

GenerateRecordType

The GenerateRecordType property determines whether the class record is generated. (Default = Checked)

HasUnknownDiscriminant

The HasUnknownDiscriminant property determines whether an unknown discriminant >) is generated for this class. (Default = False)

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces. (Default = empty string)>

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = Empty MultiLine)

ImplementationPragmas

The ImplementationPragmas property specifies the user-defined pragmas to generate in the body. (Default = Empty MultiLine)

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body. (Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

In

The In property specifies how code is generated when the type is used with an argument that has the modifier "In". When a class is used with the "In" modifier, the default is "final $type" in J.

InitCleanUpRelations

The InitCleanUpRelations property specifies whether to generate initRelations() and cleanUpRelations() operations for sets of related global instances. This property applies only to composites and global relations. (Default = Checked)

InitializationCode

The InitializationCode property adds the specified initialization code in the body of the class. A non-abstract class can have initialization code that is executed during elaboration of the associated package. (Empty MultiLine)

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the

modifier "InOut. When a class is used with the "InOut" modifier, the default is "$type" in J.

InstanceDeclaration

The InstanceDeclaration property specifies how instances are declared in code. The default value for C is as follows: struct $cname$suffix

In the generated code, the variable $cname is replaced with the object (or object type) name. The variable $suffix is replaced with the type suffix "_t," if the object is of implicit type. The default value for C++ is as follows: $cname$suffix

IsCompletedOperation

The IsCompletedOperation specifies whether state_IS_COMPLETED operations are generated as functions or macros (using #define). The possible values are as follows:

- Plain - state_IS_COMPLETED operations are generated as functions (pre-V4.2 behavior). This is the default value.
- Inline - state_IS_COMPLETED operations are generated using #define macros, if the body contains only a return statement.

(Default = Plain)

IsInOperation

The IsInOperation specifies how state_IN methods are generated.

IsLimited

The IsLimited property determines whether the class or record type is generated as limited. (Default = False)

IsNested

The IsNested property specifies whether to generate the class or package as nested. (Default = False)

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private. (Default = False)

IsReactiveInterface

The IsReactiveInterface property modifies the way reactive classes are generated. It has the following effects:

- Virtual inheritance from OMReactive
- Prevents instrumentation
- Prevents the thread argument and the initialization code (setting the active context) in the class constructor
- Creates a pure-virtual destructor (by default)

This property affects only classes that declare themselves as interfaces by having a stereotype with a name that contains the word “interface” (case-insensitive). In previous versions of Rational Rhapsody, a class could inherit from a single reactive class only, regardless of whether it was an interface or implementation class. Beginning with Version 4.0.1 MR2, a class can inherit (implement) several reactive interfaces. In Rational Rhapsody Developer for C++, you must explicitly designate reactive interfaces because the code generator applies special translation rules involving multiple inheritance from the Rational Rhapsody framework. You can designate a reactive interface in two ways:

- Set the property CPP_CG::Class::IsReactiveInterface to true.
- Use the predefined stereotype Reactive_interface. This stereotype uses stereotype-based code generation in order to automatically apply the correct property value.

Alternatively, you can define another stereotype (such as PortSpec) that sets IsReactiveInterface to true and use that stereotype. A class is considered reactive if it meets all the following conditions:

- The CPP_CG::Framework::ReactiveBase property is not empty.
- The CPP_CG::Framework::ReactiveBaseUsage property is set to true.
- One or more of the following conditions are true:
- The class has a statechart or activity diagram.
- The class is a composite class.
- The class has event receptions or triggered operations.

(Default = Checked)

Rational Rhapsody Developer for Java Note the following:

- A class is considered a reactive instance when it has an interface (for example, the Interface stereotype is applied) and it has event receptions or triggered operations.
- A reactive interface is implemented as an interface that extends RiJStateConcept.
- A class that implements a reactive interface is implemented like any other reactive class with the following exceptions:
- The class implements the reactive interface instead of RiJStateConcept.
- If the reactive interface has triggered operations, the triggered operations must be redefined in the concrete class.

JavaAnnotation

The property JavaAnnotation is used by the Rational Rhapsody code generator to insert Java annotations into generated code.

This property is used primarily for regenerating code that was reverse engineered. When you reverse engineer code that contains Java annotations, the value of the property JAVA_ReverseEngineering::ImplementationTrait::ImportJavaAnnotation determines how Rhapsody

handles the annotation code. If the value of this property is set to Verbatim, then Rational Rhapsody does not import annotations as model elements. Rather, the annotation code is stored as the value of the property JavaAnnotation. When code is later regenerated, it will include the code that was stored in this property.

Default = Blank

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

MaximumPendingEvents

The MaximumPendingEvents property specifies the maximum number of events that can be simultaneously pending in the event queue of the active class. The possible values are as follows:

- –1 - Memory is dynamically allocated.
- Positive integer - Specifies the maximum number of events.

Default = –1

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package. (Default = Public)

ObjectTypeAsSingleton

The ObjectTypeAsSingleton property enables you to generate singleton code for object-types and actors. This functionality enables you to save a singleton-type (actor) in its own repository unit, and manage that unit using a configuration management tool. Set this property for a single object-type or higher. An object-type is generated as a singleton when all of the following conditions are met:

- The object-type has the «Singleton» stereotype.
- There is one and only one object of the object-type and the object multiplicity is 1.
- The ObjectTypeAsSingleton property is set to True.

Note that when you expose a singleton object (for example, by creating a singleton object-type), Rational Rhapsody also modifies the code generated for the singleton. (Default = False)

OptimizeStatechartsWithoutEventsMemoryAllocation

The OptimizeStatechartsWithoutEventsMemoryAllocation property determines whether the generated code uses dynamic memory allocation for statecharts that use only triggered operations. (Default = False)

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out". The following table lists how classes are mapped as code when used with the Out modifier.

When a class is used with the "Out" modifier, the default is "$type" in J.

ReactiveThreadSettingPolicy

The ReactiveThreadSettingPolicy property enables you to specify how threads are set for reactive classes. The possible values are as follows:

- Default - During code generation, Rational Rhapsody adds a thread argument to the constructor.
- MainThread - Rational Rhapsody does not add an argument; the thread is set to the main thread.
- UserDefined - Rational Rhapsody does not add an argument; you must set the value for the thread yourself.

Default = Default

RecordTypeName

The RecordTypeName property specifies the name of the class record type. If this is not set, Rational Rhapsody uses class_name>_t. (Default = empty string)

RelativeEventDataRecordTypeComponentsNaming

The RelativeEventDataRecordTypeComponentsNaming property enables relative naming of event data record type components that represent events and triggered operation parameters. If this is True, no events or triggered operations will share argument names because they would generate record components with the same name (which would not compile). (Default = False)

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type. When a class is used with the "ReturnType" modifier, the default is "$type" in J.

SingletonExposeThis

The SingletonExposeThis property, when set to False, specifies that all non-static methods are considered as static methods and will not have a this parameter passed in. (Default = False)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification. (Default = Empty MultiLine)

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification. (Default = Empty MultiLine)

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you could add the @Deprecated annotation for an element by entering @Deprecated and a new line as the value of this property.

Default = Blank

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces.

Default = Empty string

Static

The property Static allows you to specify that an inner class should be declared as static. This allows you to instantiate the class outside the context of an object of the outer class.

Note that if you set the value of this property to True for a top-level class (a non-inner class), it will not affect the declaration generated for that class.

Default = Cleared

TaskBody

The TaskBody property enables you to define an alternate task body for Ada Task and Ada Task Type classes. (Default = empty string)

TriggerArgument

The TriggerArgument property specfies how the type should be passed in when used as an argument for events\triggered operations. By default, classes that are used as an argument are passed via a pointer. There are 4 other properties that effect how types are passed into and returned by regular operations: "in", "out", "in/out" and "return." See also:

- In
- InOut
- Out

Default = $type

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

See “Visibility” for more information.

Default = Public

Component

The Component metaclass contains properties that affect the Java component.

InitializationScheme

Default = ByPackage

Configuration

The Configuration metaclass contains properties that affect the configuration.

ClassStateDeclaration

The ClassStateDeclaration property supports C compilers that cannot handle enum declarations inside struct declaration. The possible values are as follows:

- InClassDeclaration - Generate the reactive statechart enum declaration in the class declaration (as in Rational Rhapsody 3.0.1).
- BeforeClassDeclaration - Generate the reactive class statechart enum declaration before the declaration of the class.

(Default = InClassDeclaration)

CodeGeneratorTool

The CodeGeneratorTool property specifies which code generation tool to use for the given configuration. The possible values are as follows:

- External - Use the registered, external code generator.

- Internal - Use the Rational Rhapsody internal code generator.

The default value is Internal.

ContainerSet

The ContainerSet property specifies the container set used to implement relations.

The possible Java values are as follows: Java(1.1)Containers Java(1.2)Containers Java(1.5)Containers (Default)

DefaultActiveGeneration

The DefaultActiveGeneration property specifies whether the default active class is created, as well as the classes for which it acts as the active context. The possible values are as follows:

- Disable - The default active singleton is not created.
- ReactiveWithoutContext - The default active singleton is created if there are reactive classes that consume events and do not have an active context explicitly specified. The default active singleton can handle only these classes.
- All - The default active singleton is generated if there is at least one event-consuming reactive class and the active singleton can handle all reactive classes that consume events - even those reactive classes that specify another active class as their active context.

(Default = ReactiveWithoutContext)

DefaultImplementationDirectory

The DefaultImplementationDirectory property specifies the relative path to the default directory for generated implementation files. The value of this property is added after the configuration path. Consider the following case:

- File C.cpp is an implementation of class C mapped to a folder Foo.
- The active configuration (cfg) is under component cmp.
- DefaultImplementationDirectory is set to “src”

Rhapsody generates C.cpp to root>\cmp\cfg\src\Foo. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

(Default = empty string)

DefaultSpecificationDirectory

The DefaultSpecificationDirectory property specifies the relative path to the default directory for generated specification files. The value of this property is added after the configuration path. Consider the following case:

- File B.h is a specification of class B that is not mapped to any file.
- The active configuration (cfg) is under component cmp.
- DefaultSpecificationDirectory is set to “inc”

Rhapsody generates B.h to root>\cmp\cfg\inc. Note the following limitations:

- This feature is not supported in COM- or CORBA-related components (C++ only).
- The predefined OSE environments (OsePPCDiab and OseSfk) are not supported due to makefile flexibility issues.
- This feature is not supported by the INTEGRITY adapter build file generator.

(Default = empty string)

DependencyRuleScheme

The DependencyRuleScheme property specifies how dependency rules should be generated in the makefile. The possible values are as follows:

- Basic - Generates only the local implementation and specification files in the dependency rule in the makefile.
- ByScope - In addition to generating the same files as the Basic option, generates the specification files of related elements (dependencies, associations, generalizations, and so on) that are in the scope of the active component.
- This option corresponds to the Rational Rhapsody 5.0.1 behavior.
- Extended - In addition to generating the same files as the ByScope option, generates the specification files of related external elements (specified using the properties CG::Class/Package::UseAsExternal) and elements that are not in the scope of the active component.

(Default = ByScope)

DescriptionBeginLine

This property enables you to specify the prefix for the beginning of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation. This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected. The following table lists the default value for each language.

Language Edition Default Value C "//" C++ ""

When you set this property, you should check the value of the lang_CG::DiffDelimiter property - if the same prefix is used, Rational Rhapsody will not update the generated code when the description is modified. If both DescriptionBeginLine and DiffDelimiter use the same prefix, modify the values of the following properties under C_CG::File:

DiffDelimiter ImplementationHeader SpecificationHeader

DescriptionEndLine

This property enables you to specify the prefix for the end of comment lines in the generated code. This functionality enables you to use a documentation system (such as Doxygen), which looks for a certain prefix to produce the documentation. This property affects only the code generated for descriptions of model elements; other auto-generated comments are not affected. The following table lists the default value for each language.

Language Edition Default Value C "/*" C++ "*/"

EmptyArgumentListName

The EmptyArgumentListName specifies the string generated for the argument list when an operation has no arguments. For example, if you set this value to “void”, for an operation foo that has no arguments, Rational Rhapsody generates the following code:

int foo (void){...}

(Default = empty string)

Environment

This property determines the target environment for a configuration. Generated code is targeted for that environment. See the Release Notes for the environments supported by Rhapsody "out-of-the-box." "Out-of-the-box" support means that Rational Rhapsody includes a set of preconfigured code generation properties for the environment and precompiled versions of the relevant OXF libraries. The precompiled OXF libraries have been fully tested.

You can also add new environments, for example if you want to generate code for another RTOS. This involves retargeting the OS wrapper files in the Rational Rhapsody framework and creating a new set of code generation properties for the target environment.

Default = JDK

ExternalGenerationTimeout

The ExternalGenerationTimeout property specifies how long, in seconds, Rational Rhapsody waits for the each class in the configuration scope to complete so you can once again make changes to the model. This property applies to both the full-featured external generator and makefile generator. For example, if you set this property to 2 and you have 10 classes, Rational Rhapsody sets a timeout of 20. If the external code generator does not complete generation in this timeframe, Rational Rhapsody displays a message in the output window saying that the generator is not responding, and you are allowed to make changes to the model. If you set this property to 0, Rational Rhapsody will not time out the generation session, and waits for the code generator to complete its task - even if it takes forever. Rhapsody waits for a notification from the full-featured external code generator, or for the process termination of a makefile generator. (Default = 0)

ExternalGeneratorFileMappingRules

The ExternalGeneratorFileMappingRules property specifies whether the external code generator uses the same file mapping and naming scheme (mapping rules) as Rational Rhapsody. If the mapping rules are different , the external generator must implement handlers to the GetFileName, GetMainFileName, and GetMakefileName events that Rational Rhapsody runs to get a requested file name and path. The possible values are as follows:

- AsRhapsody - The external generator uses the same mapping rules as Rational Rhapsody.
- DefinedByGenerator - The external generator has its own mapping rules.

The default value is AsRhapsody.

GenerateAnnotationsForNonSPARKConfigurations

The GenerateAnnotationsForNonSPARKConfigurations property specifies whether (Default = False)

GenerateDirectoryPerModelComponent

The GenerateDirectoryPerModelComponent property specifies whether to generate a separate directory for each package in the component. The possible values are as follows:

- Checked - Rational Rhapsody creates a separate directory for each package in the component.
- Cleared - A separate directory is not created for each package.

Default = Checked

GeneratorExtraPropertyFiles

The GeneratorExtraPropertyFiles property launches the default Text Editor allowing the user to edit the $OMROOT\CodeGenerator\GenerationRules\LangC\RiC_CG.ini file.

GeneratorRulesSet

The GeneratorRulesSet property enables you to specify your own rules set.

Default = Empty MultiLine

GeneratorScenarioName

The GeneratorScenarioName property specifies the scenario name for the rule, if you write your own set of code generation rules.

Default = Empty string

GenericEventHandling

The GenericEventHandling property is a Boolean value that determines whether to generate generic event-handling code. This property supports large-scale collaboration, where you might not be aware of which classes consume a base event of your part in the event hierarchy, and might not have access to parts of the model that use base events.

The framework base event class includes a new, virtual method that checks the event ID against the specified ID, thereby supplying a generic mechanism for events without super events.

For Java, the specific method is as follows: boolean isTypeOf(long id) {return lId == id;}

Each generated event that has a super event will override the method to check the ID against its own ID, then calls its base event directly to continue the check. An event without a base event will return Cleared if the ID does not equal its own. When you set the GenericEventHandling property to Cleared, event consumption code is generated as in version 3.0.1. Setting this property affects only the way events are consumed - the override on the isTypeOf() method is still generated, to allow handling of events in components that use the generic event handling. To support complete generic event handling, you should regenerate the code for all events and reactive classes.

Default = Checked

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled

with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

InitializeEmbeddableObjectsByValue

The InitializeEmbeddableObjectsByValue property specifies whether embeddable classes and object types selected in the configuration initial instances list should be allocated by value in the main() routine. (Default = False)

JarFileGenerate

Boolean property that determines whether or not a JAR file is generated as part of the build process. The value of this property is controlled by the "Generate JAR File" option on the Settings tab of the Features dialog for configurations.

Default = Cleared

JarFileGeneratorCommand

Specifies the jar command that should be carried out if the property JarFileGenerate has been set to True.

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation. (Default = Empty MultiLine)

MainFunctionArgList

This property provides a list of the main function arguments.

Default = String[] args

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you

insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters \n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

SourceListFile

The SourceListFile property specifies the name of the file containing a list of .java source files to be compiled with javac. The batch file used by the Build command (jdkmake.bat) can use the following call, rather than including a long list of source files: javac –g @files.lst This same command is generated from the following line in the MakeFileContent property for Java: javac –g @$SourceListFile If the SourceListFile property is empty, $SourceListFile is replaced with a string containing all source file names, separated by spaces (for example, "A.java B.java"). This means that if the MakeFileContent default value is not changed, you will get: javac –g @A.java B.java … If you do not want to use the file containing the list of sources, you must also change the MakeFileContent property to replace "javac –g @$SourceListFile" with "javac –g $SourceListFile".

Default = files.lst

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you could add the @Deprecated annotation for an element by entering @Deprecated and a new line as the value of this property.

Default = Blank

Dependency

The Dependency metaclass controls the dependency for a package that defines a namespace.

CreateUseStatement

The CreateUseStatement property determines whether a use statement is added to the code after the with statement. The supplier of the dependency must be a class or type. (Default = False)

GenerateOriginComment

When set to True, generates a comment before #include statements indicating which element "caused" the #include.

GeneratePragmaElaborate

The GeneratePragmaElaborate property determines whether to generate an elaborate pragma for the supplier class in the client class or package. (Default = False)

GeneratePragmaElaborateAll

The GeneratePragmaElaborateAll property determines whether to generate a pre-elaborate pragma for the supplier class in the client class or package. (Default = False)

GenerateWithClause

The GenerateWithClause property determines whether with clauses are generated for Usage dependencies. For example, you can generate a with clause for a package, P1, in the specification of another package, P2, using a dependency, D1, and generate a use clause for P1 in the body of P2. In addition, this functionality is useful for modeling inherited annotations across classes and packages. (Default = Checked)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma

statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

IncludeStyle

The IncludeStyle property controls the style of #include statements. Using this property, you can control the style of a specific dependency, or the entire configuration/component/project. To set the style for include files that are synthesized based on associations between model elements (for example, setting the type of some attribute to a class), add a «Usage» dependency between the elements and set this property to the appropriate value. The possible values are as follows:

- Default - Use angle brackets for include statements for external elements, and quotes for include statements for other elements.
- Quotes - Enclose include files in quotation marks. For example: #include "A.h"
- When a compiler encounters an include file in quotes, it searches for the file in both the current directory and the directories specified in the include path. Note that the specific algorithm used is compiler-dependent.
- AngledBrackets - Enclose include files in angle brackets. For example: #include A.h
- When a compiler encounters an include file in angle brackets, it searches for the file only in the

directories specified in the include path.

- If you set the property to AngledBrackets at the configuration level, you must also change the CG::File::IncludeScheme property to RelativeToConfiguration to ensure successful compilation.

(Default = Default)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you could add the @Deprecated annotation for an element by entering @Deprecated and a new line as the value of this property.

Default = Blank

Static

The property Static allows you to specify that a dependency should be generated as a static import.

When you apply the StaticImport stereotype to a dependency, the value of this property is set to True.

Default = Cleared

UseNameSpace

The UseNameSpace property allows you to model namespace usage. When you set a dependency to a package that defines a namespace and set this property to True, Rational Rhapsody generates a "using namespace" statement to the package namespace. (Default = False)

Event

The Event metaclass contains properties that control events.

AnimInstanceCreate

The AnimInstanceCreate property affects event creation. If you set the C_CG::Event::NoDynamicAllocAnimCreate property to False, Rational Rhapsody does not generate the event creation method, effectively disabling the ability to inject the event in animation. To enable the injection of the event, you can specify a different method to obtain an instance of the event by setting this property to the name of the method to use. (Default = empty string)

DeclarationModifier

The DeclarationModifier property enables you to add a string to the class or event declaration. The string appears between the class keyword and the class name in the generated code. For example, for a class A, the DeclarationModifier would appear as follows: class DeclarationModifier> A {…}; This property enables you to add a modifier to the class declaration. For example, if you have a class myExportableClass that is exported from a DLL using the MYDLL_API macro, you can set the DeclarationModifier property to "MYDLL_API." The generated code would then be as follows: class MYDLL_API myExportableClass { …}; This property supports two keywords: $component and $class. (Default = empty string)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name

- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

(Default = empty string)

In

The property In determines the exact syntax used when an event is used as an "in" parameter for an operation.

Default = final $type

InOut

The property InOut determines the exact syntax used when an event is used as an "in/out" parameter for an operation.

Default = $type

Out

The property Out determines the exact syntax used when an event is used as an "out" parameter for an operation.

Default = $type

ReturnType

The property ReturnType determines the exact syntax used when an event is used as the return type of an operation.

Default = $type

File

The File metaclass contains properties that control the generated code files.

DiffDelimiter

The DiffDelimiter property defines a symbol that is used to avoid overwriting an unchanged line of code during code generation. Use this property to avoid touching the source code file when the "diff-delimited" line has not changed. In general, fewer source files need to be recompiled if fewer source files are touched. For example, the DiffDelimiter symbol "//!" is used in the CPP_CG::File::Header property. This symbol is at the beginning of a line of code that includes the current code generation date. The code generator compares the code it would normally generate for that line (the current code generation date) to that previously generated (the last code generation date). If the date has not changed, the line is not overwritten, possibly preventing the file's modification time from changing (being "touched").

Default = //!

Footer

The Footer property specifies a multiline footer that is added to the end of generated Java files.

Default =

"/********** File Path: $FullCodeGeneratedFileName **********/"

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.

- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified element’s tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is "//!".

Header

The Header property specifies a multiline header that is added to the top of all generated Java files.

Default =

/********** Rhapsody : $RhapsodyVersion Login : $Login Component : $ComponentName Configuration : $ConfigurationName Model Element : $FullModelElementName //! Generated Date : $CodeGeneratedDate File Path : $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified element’s tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is "//!".

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated for the Implementation Epilog metaclass.

Leading Linefeed Added? Generated Inside or Outside or Namespace? Class Yes Outside Package No Outside

(Default = Empty MultiLine)

ImplementationFooter

The ImplementationFooter property specifies the multiline footer to be generated at the end of implementation files. The default footer template for Ada is an empty MultiLine; the default for C and C++ is as follows:

/********** File Path: $FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.

- $Tag - The value of the specified element’s tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

ImplementationHeader

The ImplementationHeader property specifies the multiline header that is generated at the beginning of implementation files. The default header template for Ada is an empty MultiLine; the default for C and C++ is as follows:

/********** Rhapsody: $RhapsodyVersion Login: $Login Component: $ComponentName Configuration: $ConfigurationName Model Element: $FullModelElementName //! Generated Date: $CodeGeneratedDate File Path: $FullCodeGeneratedFileName **********/

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified element’s tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the CPP_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated for the metaclasses.

Metaclass Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class No Outside Package Yes Outside

(Default = Empty MultiLine)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code

specified in those properties.

- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationFooter

The SpecificationFooter property specifies the multiline footer to be generated at the end of specification files. The default footer template for Ada is an empty MultiLine; the default for C and C++ is as follows:

/********** File Path: $FullCodeGeneratedFileName **********/

Footer format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified element's tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed

with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

SpecificationHeader

The SpecificationHeader property specifies the multiline header to be generated at the beginning of specification files.

Header format strings can contain any of the following keywords:

- $ProjectName - The project name.
- $ComponentName - The component name.
- $ConfigurationName - The configuration name.
- $ModelElementName - The name of the element mapped to the file. If there is more than one, this is the name of the first element.
- $FullModelElementName - The name of the element mapped to the file, including the full path. If there is more than one, this is the name of the first element.
- $CodeGeneratedDate - The generation date.
- $CodeGeneratedTime - The generation time.
- $RhapsodyVersion - The version of Rational Rhapsody that generated the file.
- $Login - The user who generated the file.
- $CodeGeneratedFileName - The name of the generated file.
- $FullCodeGeneratedFileName - The full file name.
- $Tag - The value of the specified element’s tag.
- $Property - The value of the element property with the specified name.

To avoid redundant compilation, Rational Rhapsody avoids unnecessary changes to specific lines prefixed with a special string, defined by the property CPP_CG::File::DiffDelimiter. The default DiffDelimiter value is “//!”. The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Property keywords
- Tag keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property CPP_CG::Configuration::DescriptionEndLine.

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class Yes No Inside Package Yes Yes Inside

(Default = Empty MultiLine)

Framework

The Framework metaclass contains properties that affect the Rational Rhapsody framework.

ActivateFrameworkDefaultEventLoop

The ActivateFrameworkDefaultEventLoop property specifies the framework call that initializes the framework main event loop. (Default = OXF::start($Fork);) The value of $Fork is calculated from the property CG::Configuration::StartFrameworkInMainThread for regular applications and from the property CORBA::Configuration::StartFrameworkInMainThread for CORBA servers. This property can be set at the configuration level or higher.

ActiveBase

The ActiveBase property specifies the superclass from which to specialize all threads, if the ActiveBaseUsage property is set to Checked.

Default = RiJThread

ActiveBaseUsage

The ActiveBaseUsage property specifies whether to use the superclass specified by the ActiveBase property as the superclass for all threads.

Default = Checked

ActiveDestructorGuard

The ActiveDestructorGuard property specifies the macro that starts protection for an active user object destructor. (Default = START_DTOR_THREAD_GUARDED_SECTION)

ActiveExecuteOperationName

The ActiveExecuteOperationName property sets the user object virtual table for an active object and passes it to a task in the task initialization function (RiCTask_init()). Follow these steps:

- Create a method with the following signature: struct RiCReactive * operation name> (RiCTask * const)
- Set the operation name in the ActiveExecuteOperationName property.
- Start the execution of the active object task by calling the RICTASK_START() macro on the object.

The virtual function table member name is stored in the ActiveVtblName property. (Default = empty string)

ActiveGuardInitialization

The ActiveGuardInitialization property specifies the call that makes the active object event dispatching guarded. (Default = SetToGuardThread)

ActiveIncludeFiles

The ActiveIncludeFiles property specifies the base class for threads when using selective framework includes. If a class is active and this property is defined, the file specified by the property is included in the class specification file. The default value for C++ is as follows: oxf/omthread.h The default value for C is as follows: oxf/RiCTask.h

ActiveInit

The ActiveInit property specifies the format of the declaration generated for the initializer for an active class. The default value for Ada is an empty string.

Default = m_thread = new $base("$class");

ActiveMessageQueueSize

The ActiveMessageQueueSize property specifies the size of the message queue allocated for active objects, if the ActiveMessageQueueSize property for classes is left blank.

Default = Empty string

ActiveStackSize

The ActiveStackSize property specifies the size of the stack allocated for active objects, if the ActiveStackSize property for classes is left blank.

Default = Empty string

ActiveThreadName

The ActiveThreadName property specifies the name of threads, if the ActiveThreadName property for classes is left blank.

Default = ""

ActiveThreadPriority

The ActiveThreadPriority priority specifies the priority of threads, if the ActiveThreadPriority property for classes is left blank.

Default = Empty string

ActiveVtblName

The ActiveVtblName property stores the name of the virtual function table associated with a task (the RiCTask member of the structure). (Default = $objectName_activeVtbl)

BooleanType

The BooleanType property specifies the Boolean type used by the framework. (Default = bool)

CurrentEventId

The CurrentEventId property specifies the call or macro used to obtain the ID of the currently consumed event. (Default = OM_CURRENT_EVENT_ID)

DefaultProvidedInterfaceName

The DefaultProvidedInterfaceName property specifies the interface that must be implemented by the "in" part of a rapid port. See the Rational Rhapsody Help for more information on rapid ports.

Default = DefaultProvidedInterface

DefaultReactivePortBase

The DefaultReactivePortBase property stores the base class for the generic rapid port (or default reactive port). This base class relays all events. See the Rational Rhapsody Help for more information on rapid ports.

Default = RiJDefaultReactivePort

DefaultReactivePortIncludeFiles

The DefaultReactivePortIncludeFiles property specifies the include files that are referenced in the generated file that implements the class with the rapid ports. See the Rational Rhapsody Help for more information on rapid ports.

Default = oxf/OMDefaultReactivePort.h

DefaultRequiredInterfaceName

The DefaultRequiredInterfaceName property specifies the interface that must be implemented by the "out" part of a rapid port. See the Rational Rhapsody Help for more information on rapid ports.

Default = DefaultRequiredInterface

EnableDirectReactiveDeletion

The EnableDirectReactiveDeletion property specifies the call to the framework that supports direct deletion of reactive instances (using the delete operator) instead of graceful framework termination (using the reactive destroy() method). When using destroy(), the object waits in a zombie mode until all the events that are designated to it are removed from the active context queue, and then self -destructs. In this scheme, there is no need to traverse the queue of the active context to cancel pending events, and there is no need to make the reactive destructor guarded to ensure safe deletion. A reactive object can be either in a graceful termination or forced deletion (using the delete operator) state: you cannot use graceful deletion on an object that allows forced deletion, and vice versa. You can set a single reactive object in a forced deletion state, or set the entire system (all reactive instances) in a forced deletion state (as is done for backward compatibility). Graceful termination should not be used when a reactive part (of a composite class) runs in a context of an active object that is not part of, and different from, the composite active context. If you are using a Rhapsody library component as part of an application where the main is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init(). If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add

independent, backward-compatibility activation calls prior to the initialize() call. Note that the property CPP_CG::Framework::UseDirectReactiveDeletion must be set to True for this property to take effect. When it is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init(). (Default = OXF::supportExplicitReactiveDeletion();)

EventBase

The EventBase property specifies the base class for all events, if the EventBaseUsage property is set to Checked.

Default = RiJEvent

EventBaseUsage

The EventBaseUsage property specifies whether to use the event superclass specified by the EventBase property as the parent of all events.

Default = Checked

EventGenerationPattern

The EventGenerationPattern property supplies some of the information needed to generate code for Send Action elements in C, C++, and Java.

For each language, code generation for this element is determined by the following properties:

- C_CG::Framework::EventGenerationPattern - general format
- CG::Framework::EventToPortGenerationPattern - used when sending even to a port

Default = $target$(goArr)gen(new $event)

Note: Rhapsody does not support roundtripping for Send Action elements.

EventIncludeFiles

The EventIncludeFiles property specifies the base class for events when using selective framework includes. If events are defined in a package, the file specified by this property is included in the package specification file to enable the use of events and timeouts in the package. The default value for C is as follows: oxf/RiCEvent.h The default value for C++ is as follows: oxf/event.h

EventSetParamsStatement

The EventSetParamsStatement property specifies a template for the body of the setParams() method, provided by the Rational Rhapsody framework for Java, to set the parameters of an event. For example, for an event of type evOn(), the default template would generate the following code in the body of the setParams() method: evOn params = (evOn) event;

Default = $eventType params = ($eventType) event;

FrameworkInitialization

The FrameworkInitialization property specifies the framework initialization code that is called by the main. The default value is as follows: OXF::initialize($(Argc)$(Argv)$(AnimationPortNumber) $(RemoteHost)$(TimerResolution)$(TimerMaxTimeouts) $(TimeModel)))

HeaderFile

The HeaderFile property specifies the framework header files to be included in objects that are within the scope of a particular configuration.

To optimize your code for size, leave the HeaderFile property blank. In this way, you can explicitly include the framework only when needed.

Default = Blank

IncludeHeaderFile

The IncludeHeaderFile property specifies whether to include the framework header files specified by the CG::Framework::HeaderFile property in the project.

Default = Checked

InnerReactiveClassName

The InnerReactiveClassName property enables you to specify the name of a reactive class that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive. The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance.

Default = Reactive

InnerReactiveInstanceName

The InnerReactiveInstanceName property enables you to specify the name of a reactive instance that serves as a bridge between a reactive class in your model and the framework. The implementation scheme of reactive classes is different in Java than in C++. Java does not allow inheritance from the reactive framework classes because that would mean that you would not be able to inherit from an additional base class that might not be reactive. The chosen alternative is to delegate an inner class instance that inherits from RiJStateReactive. Delegation is the implementation of an interface that forwards relevant messages to the inner class instance.

Default = reactive

InstrumentVtblName

The InstrumentVtblName property specifies the name of the virtual function table associated with animation objects. Each animated object has its own virtual function table (Vtbl). This table enables you to create your own framework, with its own virtual instrumentation functions, and connect it to Rational Rhapsody. (Default = $objectName_instrumentVtbl)

IsCompletedCall

The IsCompletedCall property specifies the call or macro that determines whether the state reached a final state so it can be exited on a null transition. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name. (Default = IS_COMPLETED($State))

IsInCall

The IsInCall property specifies the query that determines whether the state is in the current active configuration. The property supports the $State keyword so you can use state-based calls. The keyword is resolved to the state implementation (code) name. (Default = IS_IN($State))

MakeFileName

The MakeFileName property enables you to specify a new name for the makefile. To use this property, add the following line to the .prp file:

Property MakeFileName String "MyFileName"

In this syntax, MyFileName specifies the name of the makefile.

NullTransitionId

The NullTransitionId property specifies the ID reserved for null transition consumption. (Default = OMEventNullId)

OperationGuard

The OperationGuard property specifies the macro that guards an operation. (Default = GUARD_OPERATION)

ProtectedBase

The ProtectedBase property specifies the base class for protected objects, if the ProtectedBaseUsage

property is set to Checked.

Default = Empty string

ProtectedBaseUsage

The ProtectedBaseUsage property specifies whether to use the class specified by the ProtectedBase property as the base class for protected objects.

Default = Cleared

ProtectedClassDeclaration

The ProtectedClassDeclaration property affects how protected classes are implemented. Beginning with Rational Rhapsody 4.0, instead of inheriting from OMProtected, the class embeds an aggregate OMProtected. The aggregate member and helper methods are defined in the macro OMDECLARE_GUARDED (defined in omprotected.h). (Default = OMDECLARE_GUARDED)

ProtectedIncludeFiles

The ProtectedIncludeFiles property specifies the base class for protected classes when using selective framework includes. The default value for C is as follows: oxf/RiCProtected.h The default value for C++ is as follows: oxf/omprotected.h

ProtectedInit

The ProtectedInit property specifies the declaration generated for the initializer for guarded objects. The default value for Ada is an empty string. The default value for C is as follows: $base_init($member)

ReactiveBase

The ReactiveBase property specifies the base class for all reactive classes, if the ReactiveBaseUsage property is set to Checked.

Default = RiJStateReactive

ReactiveBaseUsage

The ReactiveBaseUsage property specifies whether to use the class specified by the ReactiveBase property as the base class for all reactive objects.

Default = Checked

ReactiveConsumeEventOperationName

The ReactiveConsumeEventOperationName property sets the user object virtual table for a reactive object. Follow these steps:

- Create a method with the following signature: void operation name>(RiCReactive * const, RiCEvent*)
- Set the operation name in the ReactiveConsumeEventOperationName property.

Rational Rhapsody Developer for Ada ignores all the values for the properties under the Framework metaclass except for this one. (Default = empty string)

ReactiveCtorActiveArgDefaultValue

The ReactiveCtorActiveArgDefaultValue property specifies the default value of the active context argument in a reactive constructor. (Default = 0)

ReactiveCtorActiveArgName

The ReactiveCtorActiveArgDefaultValue property specifies the name of the active context argument in a reactive constructor. (Default = activeContext)

ReactiveCtorActiveArgType

The ReactiveCtorActiveArgDefaultValue property specifies the type of the active context argument in a reactive constructor. (Default = IOxfActive)*

ReactiveDestructorGuard

The ReactiveDestructorGuard property specifies the macro that starts protection of a section of code used for destruction of a reactive instance. This prevents a "race" (between the deletion and event dispatching) when deleting an active instance. (Default = START_DTOR_REACTIVE_GUARDED_SECTION)

ReactiveEnableAccessEventData

The ReactiveEnableAccessEventData property specifies the code to be used to enable access to the specific event data in a transition (typically by assigning a local variable of the appropriate type). The property supports the $Event keyword so you can specify the event type. (Default = OMSETPARAMS($Event);)

ReactiveGuardInitialization

The ReactiveDestructorGuard property specifies the framework call that makes the event consumption of a specific reactive class guarded. (Default = setToGuardReactive)

ReactiveHandleEventNotConsumed

The ReactiveHandleEventNotConsumed property registers a method to handle unconsumed events in a reactive class. Specify the method name as this property's value. (Default = empty string)

ReactiveHandleTONotConsumed

The ReactiveHandleTONotConsumed property registers a method to handle unconsumed trigger operations in a reactive class. Specify the method name as this property's value. (Default = empty string)

ReactiveIncludeFiles

The ReactiveIncludeFiles property specifies the base classes for reactive classes when using selective framework includes. If a class is reactive and this property is defined, the file specified by the property is included in the class specification file. For reactive classes, the header files specified by the following properties are also included:

- EventIncludeFiles - For the event base class
- ActiveIncludeFiles - If the class is guarded or instrumented

The default value for C is as follows: oxf/RiCReactive.h

ReactiveInit

The ReactiveInit property specifies the declaration for the initializer generated for reactive objects. The default pattern for C is as follows: $base_init($member, (void*)$mePtr, $task, $VtblName); The $base variable is replaced with the name of the reactive object during code generation. The string "_init" is appended to the object name in the name of the operation. For example, if the reactive object is named A, the initializer generated for A is named A_init(). The $member variable is replaced with the name of the reactive member (equivalent to the base class) of the object during code generation. The $mePtr variable is replaced with the name of the user object (the value of the Me property). The member and mePtr objects are not equivalent if the user object is active. The $VtblName variable is replaced with the name of the virtual function table for an object, specified by the ReactiveVtblName property. The default value for Ada is an empty string. The default for C is as follows: $base_init($member, (void*)$mePtr, $task, $VtblName);

Default = reactive = new Reactive($task);

ReactiveInterface

The ReactiveInterface property specifies the name of the interface class that forwards messages to an inner class instance of a reactive class in order to implement its reactive behavior.

Default = RiJStateConcept

ReactiveSetEventHandlingGuard

The ReactiveSetEventHandlingGuard property enables you to control the code generated within the constructor of a reactive class. When you use this property with guarded triggered operations, it enables

guarding of the event handling (in order to provide mutual exclusion between the event and TO handling). (Default = setEventGuard(getGuard());)

ReactiveSetTask

The ReactiveSetTask property specifies the string that tells a reactive object whether it is an active or a sequential instance. The default value for Ada is an empty string. The default value for C is as follows: RiCReactive_setActive($member, $isActive); The default value for C++ is as follows: setThread($task, $isActive);

ReactiveVtblName

The ReactiveVtblName property specifies the name of the virtual function table (Vtbl) associated with a reactive object. Each reactive object has its own Vtbl, which enables you to create your own framework and connect it to Rational Rhapsody. (Default = $objectName_reactiveVtbl)

SetManagedTimeoutCanceling

The SetManagedTimeoutCanceling property is a property for backward compatibility that specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (where OMTimerManager is responsible for cancellation of timeouts) or the Rational Rhapsody 6.0 scheme. In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object). This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling). If you are using a Rhapsody library component as part of an application where the main is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to OXF::init(). If you want to remove part or all of the compatibility features, call OXF::initialize() instead of OXF::init() (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the initialize() call. (Default = OXF::setManagedTimeoutCanceling(true);)

SetRhp5CompatibilityAPI

The SetRhp5CompatibilityAPI property specifies the call that configures models created before Rhapsody 6.0 so they use the 5. x version of the framework instead of the new one. See UseRhp5CompatibilityAPI for more information on Version 5. x compatibility mode. (Default = OXF::setRhp5CompatibleAPI(true);)

StaticMemoryIncludeFiles

The StaticMemoryIncludeFiles property specifies the files to be included in the package specification file if static memory management is enabled and you are using selective framework includes. (Default = oxf/MemAlloc.h)

StaticMemoryPoolDeclaration

The StaticMemoryPoolDeclaration property specifies the declaration of the memory pool for timeouts. The default value is as follows:

DECLARE_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances)

StaticMemoryPoolImplementation

The StaticMemoryPoolImplementation property specifies the generated code in the implementation file for a memory pool implementation (see the BaseNumberOfInstances property). The default value is as follows:

IMPLEMENT_MEMORY_ALLOCATOR($Class, $BaseNumberOfInstances, $AdditionalNumberOfInstances, $ProtectStaticMemoryPool)

TestEventTypeCall

The TestEventTypeCall property specifies the test used in event consumption code to check if the currently consumed event is of a given type. (Default = IS_EVENT_TYPE_OF($Id))

TimeoutId

The TimeoutId property specifies the ID reserved for timeout events. (Default = OMTimeoutEventId)

TimerMaxTimeouts

The TimerMaxTimeouts property specifies the maximum number of timeouts allowed simultaneously in the system, if the TimerMaxTimeouts property for the configuration is not overridden. In the framework, the default number of timers is 100.

Default = Empty string

TimerResolution

The TimerResolution property specifies the length of time that must pass until the timer should check for matured timeouts. In the framework, the default number of timers is 100.

Default = Empty string

UseDirectReactiveDeletion

The UseDirectReactiveDeletion property determines whether direct deletion of reactive instances (using the delete operator) is used instead of graceful framework termination (using the reactive destroy() method). When this property is set to True, the code specified in the EnableDirectReactiveDeletion is generated in the main prior to the call to OXF::init(). See EnableDirectReactiveDeletion and the upgrade history on the support site for more information on this functionality. (Default = False)

UseManagedTimeoutCanceling

The UseManagedTimeoutCanceling property specifies whether the framework uses the pre-Rhapsody 6.0 scheme of timeout creation and cancellation (so OMTimerManager is responsible for cancellation of timeouts). In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object). This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance). The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling). When loading a pre-6.0 model, Rational Rhapsody sets the project CPP_CG::Framework::UseManagedTimeoutCanceling to True to set the system-compatibility mode. See the upgrade history on the support site for more information. (Default = False)

UseRhp5CompatibilityAPI

The UseRhp5CompatibilityAPI property specifies whether to use the virtual functions of the core implementation classes that existed in the pre-Rhapsody 6.0 framework. The Rhapsody 6.0 framework introduces a set of interfaces for the core behavioral framework. The interfaces define a concise API for the framework and enable you to replace the actual implementation of these interfaces while maintaining the framework behavior. As a result of the interfaces' introduction, the framework behavioral classes (OMReactive, OMThread, and OMEvent) use a new set of virtual operations to implement the interfaces and provide the behavioral infrastructure. To support existing customizations of these classes (made by inheriting and overriding the virtual operations), the framework can work in a mode where the pre-6.0 API virtual operations are called. When loading a pre-6.0 model, Rational Rhapsody sets the project property CPP_CG::Framework::UseRhp5CompatibilityAPI to True to set the system-compatibility mode. If this is set to True, the pre-6.0 API is called by the framework instead of the interface-based API. Without this flag, user customizations will compile but will not be called. See the upgrade history on the support site for more information on the Version 5. x compatibility mode. (Default = False)

Generalization

The Generalization metaclass contains a property used to support generalization. See the Rational Rhapsody Help for more information on generalization.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the

specific argument Animate property settings.

(Default = Checked)

JDK

The JDK metaclass contains properties that manipulate the operating system environment.

AdaptorSearchPath

The AdaptorSearchPath property specifies the path to the operating system configuration file. This path is added to the generated makefile search path. This property reflects the change in Version 4.1 where the RTOS-specific code was removed from the framework code and placed in separate files, and a new adapter builder was created. This new scheme makes it easier to add a custom adapter because you do not need to modify the framework files. To upgrade a custom adapter to the new scheme, you must do the following:

- Create the relevant operating system configuration file.
- Add the file directory to the search path in the framework makefiles.
- Add the AdaptorSearchPath property to the adapter environment properties, with the value set to the path to the operating system configuration file.

Default =

AdditionalReservedWords

The AdditionalReservedWords property is a string that enables you to specify additional reserved keywords that Rational Rhapsody will not allow you to use. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The property value is checked at runtime when you name/rename an element, based on the active configuration environment setting. Note that this property affects the algorithm only when the active configuration is of the selected environment.

Default = Empty string

BriefErrorMessages

The BriefErrorMessages property determines whether a /brief option is generated on SPARK Examiner calls. (Default = Checked)

BSP

The BSP property specifies the board support package (BSP) for the system. If you need to change the value of the CPU, you can simply reset the value in this property instead of changing the value in the MakeFileContent property. (Default = "PENTIUM")

BSP_Libraries

The BSP_Libraries property specifies the default BSP libraries to link to. The default value is as follows:

"%RAVENROOT%/bsp/raven/standard_model" "%RAVENROOT%/bsp/system/simulator" "%RAVENROOT%/lib/extensions"

BuildCommandSet

The BuildCommandSet property generates a set of commands in the makefile to build a debug or release version of the configuration. To change this property, use the Configuration window in the browser - do not change it using the Properties window or by modifying the site.prp file. Note that this property also affects the names of the framework libraries used in the link. The possible values are as follows:

- Debug - Generate the debug command set in the makefile.
- DebugNoExp - Generate the debug command set in the makefile without the exceptions flag (:cx_option=exceptions).
- Release - Generate the release command set in the makefile.
- ReleaseNoExp - Generate the release command set in the makefile without the exceptions flag (:cx_option=exceptions).

Default = Debug

BuildInIDE

The boolean property BuildInIDE allows you to specify the program that should perform the build - Rational Rhapsody or the IDE with which it is being used. If the value of the property is set to True, then Rational Rhapsody calls the IDE build command when its own build command is started.

This property corresponds to the "Build configuration in IDE" option on the IDE tab of the features dialog for configurations.

Default = True

CodeTestSettings

The CodeTestSettings property specifies the compiler command settings. This property supports integration with Applied Microsystems Corporation® CodeTest™. (Default = CXX=$AMC_HOME)\bin\ctcxx)

COM

The COM property specifies whether the current component is a COM component. By default, this property is set to True for all COM components (stereotypes COM DLL, COM EXE, and COM TLB). If you set this property in the generated makefile for the component, the linker option /SUBSYSTEM is set to :windows. (Default = False)

CompileCommand

The CompileCommand property is a string that enables you to specify a different compile command. (Default = empty string)

CompileSwitches

The CompileSwitches property specifies the compiler switches. This property replaces the CPPCompileSwitches property.

Default = Empty MultiLine

ConvertHostToIP

The ConvertHostToIP property specifies whether to convert the host name to an IP number. This is necessary because pSOSystem does not include a name service. Default = Checked

DEFExtension

The DEFExtension property is a string that specifies the extension for DLL definition files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. (Default = .def)

DllExtension

The DllExtension property is a string that specifies the extension for DLL files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. (Default = .dll)

DuplicateLibsListInMakeFile

The DuplicateLibsListInMakeFile property is a Boolean value that specifies whether Rational Rhapsody should duplicate the libraries list in the generated makefile link command. This property supports linkers that are sensitive to library order in the link command. (Default = Checked)

EntryPoint

The EntryPoint property specifies the name of the main program for a given environment.

Default = main

See also the definition of the EntryPointDeclarationModifier property for more information.

EntryPointDeclarationModifier

The EntryPointDeclarationModifier property specifies a modifier for the entry point declaration. This property allows generation of the main() function in the specified syntax. To modify the main() signature implemented in the OSE adapter, do the following:

- Add the property EntryPointDeclarationModifier to your environment properties and set it to the main return value and name. For example: "int main"
- Set the EntryPoint property to the main arguments. For example: "int a, long b, char**"
- Generate the code.

You will get the following main() declaration:

int main(int a, long b, char** c) { ... }

(Default = OS_PROCESS)

EnvironmentVarName

The EnvironmentVarName property specifies the name of the global variable that you must define in order to use the Embedded C++ compiler. It is used by the MultiMakefileGenerator. The value replaces the $EnvironmentVarName value> keyword inside the property value BLDAdditionalOptions. (Default = INTEGRITY_ROOT)

ErrorMessageTokensFormat

The ErrorMessageTokensFormat, working with the ParseErrorMessage property, specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler. ErrorMessageTokensFormat defines the number and location of tokens within the regular expression defined by the ParseErrorMessage property. ErrorMessageTokens has three parameters, each with an integer value:

- TotalNumberOfTokens - The number of tokens in the regular expression
- FileTokenPosition - The position of the file name token in the expression
- LineTokenPosition - The position of the line number token in the expression

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ESTLCompliance

The ESTLCompliance property is a Boolean value that determines whether you are using the Embedded C++ (ESTL) environment and conform to its requirements. In instrumentation mode, the Rational Rhapsody code generator usually creates an OMAnimatedUser Class> friend class for each user-defined class. This class inherits from AOMInstance, if its User Class> does not inherit from another class in the model. This inheritance is virtual and is needed for multiple inheritance support. Because ESTL does not support multiple inheritance (as far as virtual inheritance), the Rational Rhapsody Developer for C++ code generator will not create “virtual” inheritance if ESTLCompliance is set to True. To support ESTL

compliance, Rational Rhapsody includes a new check to recognize the following elements of ESTL-noncompliance:

- Multiple inheritance, caused by the user model (several superclasses)
- Multiple inheritance, caused by Rhapsody (an active reactive class is generated with two base classes: OMReactive and OMThread)
- Multiple inheritance, caused by a combination of the following factors:
- An active class containing a superclass
- A reactive class containing a superclass
- Virtual inheritance, declared by the user in the features of the superclass

In these cases, Rational Rhapsody displays the following warning message for each problematic class: "ESTL does not support multiple/virtual inheritance" Note that this check runs only when the ESTLCompliance property is set to True. (Default = Checked)

ExeExtension

The ExeExtension property specifies the extension that is appended to compiled executable components for a given environment.

Default = .class

FileDependencies

The FileDependencies property specifies which framework specification files and implementation files should be included in model elements. The file inclusions are generated in the makefile. The default value for GNAT and OsePPCDiab/OseSfk is an empty string. The file dependency string for most of the C and C++ environments is as follows: $OMSpecIncludeInElements $OMImpIncludeInElements

GeneratedAllDependencyRule

The GeneratedAllDependencyRule property specifies whether to automatically generate the “all:” rule as part of the expansion of the $OMContextMacros keyword in the makefile. If this is False, you can define the makefile macros manually. (Default = False)

GetConnectedRuntimeLibraries

The GetConnectedRuntimeLibraries property specifies the list of libraries that need to be linked with Web-enabled projects (when the Web Instrumentation check box is enabled). During code generation, these libraries are added to the generated makefile. Note that if you select Release Build Set (in the Environment Settings group on the Settings page), these libraries are automatically added with the R postfix (the Rational Rhapsody convention for framework libraries).

Default = $RhapJarsDir\webComponents.jar

HasIDEInterface

The HasIDEInterface property is a Boolean value that specifies whether IDE support is enabled. If IDE support is enabled (True), the IDEInterfaceDLL property points to an IDE adapter that provides connection to the IDE. If the property is set to False, IDE support is disabled and IDE services are not attempted. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. The default value for QNXNeutrinoCW is False; for the other environments, the default value is True.

ImpExtension

The ImpExtension property specifies the extension that Rational Rhapsody appends to generated implementation files for a given language and environment. The default values for Java is None

InvokeExecutable

The InvokeExecutable property specifies the command used to run an executable file.

Default = "$OMROOT\etc\jdkrun.bat" "$makefile" Main$ComponentName

InvokeMake

The InvokeMake property specifies the command that is started for Build. The command syntax and batch file are target-dependent. The command syntax is as follows:

"rhapsody_dir>\share/etc/Executer.exe""dir; dir | more"

The property InvokeMake may include the value of any other property in the relevant environment. To use this feature, simply include the name of the property preceded by $. As shown in the example below (from the VxWorks RTP environment), the value of the InvokeMake property includes the value of the property BSP.

"$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\vx6make.bat\" $makefile $maketarget 6.2 $BSP gnu"

Default = "$OMROOT/etc/Executer.exe" "\"$OMROOT\etc\jdkmake.bat\" $makefile $maketarget"

InvokeMakeGenerator

The InvokeMakeGenerator specifies the path to the executable for an external makefile generator. This external generator is started each time you request a makefile generation. If the specified path is incorrect, Rational Rhapsody generates an error message. If you are using a full-featured external code generator, this property setting is ignored. The default values are as follows:

Java - None

IsFileNameShort

The IsFileNameShort property specifies whether to truncate generated file names to 8.3 format. If this is Checked:

- The file name is not truncated.
- If the FileName property is not blank, its value overrides any automatic file name synthesis.
- If the file name is longer than eight characters, the Checker reports this prior to code generation.

Default = Cleared

LibExtension

The LibExtension property specifies the extension that is appended to compiled library components for a given environment.

Default = Empty string

MainIncludes

The MainIncludes property is a string that specifies the files that need to be included in the main program generated for an application. (Default = ose.h)

MakeExtension

The MakeExtension property specifies the extension that Rational Rhapsody appends to makefiles.

Default = .bat

MakeFileContent

The MakeFileContent property specifies how the makefile is generated for a configuration. The makefile can be of any length. The InvokeMake property references this makefile. A makefile has the following sections:

- Target type
- Compilation flags
- Commands definitions
- Generated macros
- Predefined macros
- Generated dependencies
- Linking instructions

The following sections describe the contents of the makefile in detail.

Target Type The target type section of the makefile contains the macros needed to build either a Debug or Release version of a configuration.

echo off set RHAP_JARS_DIR=$OMRoot\LangJava\lib set SOURCEPATH=$ConfigSources$ComponentSources%SOURCEPATH% set CLASSPATH=$ConfigClasspath$ComponentClasspath%CLASSPATH%;.;%RHAP_JARS_DIR%\oxf.jar;%RHAP_JARS_ set PATH=$ConfigPath$ComponentPath%RHAP_JARS_DIR%;%PATH%; set INSTRUMENTATION=$INSTRUMENTATION set BUILDSET=$BuildSet if %INSTRUMENTATION%==Animation goto anim :noanim set CLASSPATH=%CLASSPATH%;%RHAP_JARS_DIR%\oxfInstMock.jar goto setEnv_end :anim set CLASSPATH=%CLASSPATH%;%RHAP_JARS_DIR%\oxfInst.jar :setEnv_end if "%1" == "" goto compile if "%1" == "build" goto compile if "%1" == "clean" goto clean if "%1" == "rebuild" goto clean if "%1" == "run" goto run :clean echo cleaning class files $ClassClean if "%1" == "clean" goto end :compile if %BUILDSET%==Debug goto compile_debug echo compiling JAVA source files javac $ConfigCompilerSwitches @$SourceListFile goto end :compile_debug echo compiling JAVA source files javac -g $ConfigCompilerSwitches @$SourceListFile goto end :run java %2 :end

Java Users To generate Java JAR files, run the jar command from the makefile, using the MakeFileContent property. You can specify the manifest file as an external file with a text element in it. You can add additional files to the model for completeness. There is no specialized support for RMI in Rational Rhapsody. Call the JDK and run the relevant tools manually, or via the generated makefile (change the MakeFileContent property).

NullValue

The NullValue property enables you to specify an alternative expression for NULL in the generated code.

Default = NULL

ObjCleanCommand

The ObjCleanCommand property specifies the environment-specific command used to clean the object files generated by a previous build.

Default = if exist $OMFileObjPath del $OMFileObjPath

ObjectName

The ObjectName property specifies an alternative name for the compiled object file in the generated makefile. (Default = empty string)

ObjExtension

The ObjExtension property specifies the extension appended to compiled object files for a given environment.

Default = .class

OMCPU

The OMCPU property is resolved in the MakeFileContent property as the CPU type. The QNXNeutrinoCW environment uses the custom keywords feature to enable you to select the CPU without modifying the makefile template. (Default = x86)

OMCPU_SUFFIX

The OMCPU_SUFFIX property is resolved in the MakeFileContent property as the CPU extension (which is required for PPC targets). The QNXNeutrinoCW environment uses the custom keywords feature to enable you to select the CPU without modifying the makefile template. (Default = ($NO_CPU_EXT))

OpenHTMLReports

The OpenHTMLReports property specifies whether to open the HTML reports when the examination is complete. (Default = Checked)

OSFileSystemCaseSensitive

The OSFileSystemCaseSensitive property specifies whether the OS file system for a given environment is case sensitive.

Default = Cleared

ParseErrorDescript

The property ParseErrorDescript is used to define a regular expression that represents the format of build error messages. The property is used to extract the "description" part of the message so that it can be displayed in the Description column on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:] (.)*

ParseErrorMessage

The ParseErrorMessage property defines a regular expression that matches the format of error messages. The regular expression can contain two or more tokens. For example: (lineNumber), (fileName), and (Error|Fatal|Warning) Along with the ErrorMessageTokensFormat property, ParseErrorMessage specifies the expected format of error messages for a given environment. These two properties retrieve the file name and line number of errors reported by the compiler.

Default =([^:]+)[:]([0-9]+)[:]

ParseSeverityError

The property ParseSeverityError is used to define a regular expression that represents the format of compilation messages with severity "error". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = ([^:]+)[:]([0-9]+)[:]

ParseSeverityWarning

The property ParseSeverityWarning is used to define a regular expression that represents the format of compilation messages with severity "warning". This property is used to determine the type of icon that should be displayed alongside the message on the Build tab of the Output window.

Default = (.)(Note:|warning:) (.*)*

PathDelimiter

The PathDelimiter property specifies an alternative path separator for code generation.

Default = /

ProcessToKillAtStopExec

The ProcessToKillAtStopExec property stops the running process of the Java application when you select Code > Stop Execution in the Rational Rhapsody GUI.

Default = Java

QuoteOMROOT

The QuoteOMROOT property specifies whether to enclose the value of the OMROOT path variable in double quotes in the generated makefile.

Default = Checked

RCExtension

The RCExtension property is a string that specifies the extension for resource files. In general, this is an environment property that can be contained in any of the environment metaclasses supported by Rhapsody. (Default = .rc)

RemoteHost

The RemoteHost property specifies the name of the host machine when you run an application remotely. In such configurations, the host is the machine running Rhapsody, whereas the target is the machine running the application. To run remotely, the UseRemoteHost property must be set to True. If

UseRemoteHost is True and RemoteHost is blank, the current host name is used for the remote host. You can use this as a workaround if you have problems running animated applications on Windows 95. The RemoteHost property can be left blank if both the application and Rhapsody are running on the same machine.

Default = Empty string

SpecExtension

The SpecExtension property determines the extension that Rational Rhapsody appends to generated specification (header) files for a given language and environment.

Default = .java

SpecFilesInDependencyRules

The SpecFilesInDependencyRules property specifies whether to include specification files in makefile dependency rules. The OSE makefile does not support specification files in the Dependency line. Therefore, the default for OSE is False. When this property is False, no .h files are added to the Dependency line of the makefile. The default value for GNAT is True; for OSE, the default value is False.

SubSystem

The SubSystem property is a string that defines the type of the program for the Microsoft linker. The possible values are as follows:

- CONSOLE - Used for a Win32 character-mode application
- WINDOWS - Used for an application that does not require a console
- NATIVE - Applies device drivers for Windows NT
- POSIX - Creates an application that runs with the POSIX subsystem in Windows NT

(Default = /SUBSYSTEM:console)

TargetConfigurationFileName

The TargetConfigurationFileName property specifies the name of the target configuration file to be passed as an argument to the SPARK Examiner. (Default = empty string)

UnixLineTerminationStyle

The UnixLineTerminationStyle property specifies whether generated files use the UNIX end-of-line style. If this property is set to Cleared, the end-of-line style depends on the host type (for example, DOS style on Windows machines, and UNIX style on Solaris machines).

Default = Cleared

UnixPathNameForOMROOT

The UnixPathNameForOMROOT property specifies whether the makefile must include UNIX-style path names. The pRISM compilers do not tolerate DOS-style path conventions. If you do not set this property correctly, there might be many compilation problems.

Default = Cleared

UseActorsCode

The UseActorsCode property specifies whether code is generated for actors. The value of the property should be synchronized with the configuration Generate Code For Actors checkmark (located in the configuration Initialization tab). (Default = False)

UseNewBuildOutputWindow

The property UseNewBuildOutputWindow determines which tab is brought to the front of the Output window after the completion of a build action. If set to True, the Build tab is shown. Otherwise, the Log tab is shown.

This property can be set individually for different environments.

If you would like to have the Log tab shown for all environments, you can set the value of the property CG::General::ShowLogViewAfterBuild to True.

Default = Checked

UseNonZeroStdInputHandle

The UseNonZeroStdInputHandle property is a Boolean value that specifies whether to use a non-zero standard input handle. For INTEGRITY, OBJECTADA, RAVEN_PPC, and SPARK the default value is False; for the other environments, the default value is True.

UseRemoteHost

The UseRemoteHost property specifies whether you intend to run an application remotely over a network by default in a given environment.

Default = Cleared

Operation

The Operation metaclass contains properties that control operations.

ActivityReferenceToAttributes

The ActivityReferenceToAttributes property specifies whether Rational Rhapsody should generate references in the functor object, thereby giving you direct access to the attributes of the class that owns the modeled operation (without the need for this_). See the section on activity diagrams in the Rational Rhapsody Help for detailed information about modeled operations and functor classes. (Default = Checked)

AnimAllowInvocation

The AnimAllowInvocation property specifies whether primitive and triggered operations can be called during instrumentation. If an operation is called during animation, its return value is displayed in the output window; if it is traced, the return value is displayed in the console. The possible values are as follows:

- All - Enable all operation calls, regardless of visibility.
- None - Do not enable operation calls.
- Public - Enable calls to public operations only.
- Protected - Enable calls to protected operations only.

(Default = None)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

(Default = empty string)

EntryCondition

The EntryCondition property specifies the task guard. (Default = empty string)

GenerateImplementation

The GenerateImplementation property specifies whether to generate the body for the operation. To generate Import pragmas in Rational Rhapsody Developer for Ada, set this property to False and add the "pragma..." declaration in the Ada_CG::Operation::SpecificationEpilog property. (Default = Checked)

ImplementActivityDiagram

The ImplementActivityDiagram property enables or disables code generation for activity diagrams. (Default = False)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

ImplementationName

The ImplementationName property enables you to give an operation one model name and generate it with another name. It is introduced as a workaround that enables you to generate const and non-const operations with the same name. For example:

- Create a class A.
- Add a non-const operation f().
- Add a const operation f_const().
- Set the CPP_CG::Operation::ImplementationName property for f_const() to “f.”
- Generate the code.

The resulting code is as follows: class A { ... void f(); / the non const f */ ... void f() const; /* actually f_const() */ ... }; The creation of two operations with the same signature, differing only in whether it is a const, is a common practice in C++, especially for STL users. (Default = empty string)*

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations
- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for Java Because inlining has no meaning in Java, the Inline property is set to none.

Default = none

IsAnimationHelper

The IsAnimationHelper property indicates whether the operation should be generated only when animating the model. (Default = False)

IsEntry

The IsEntry property indicates whether the operation is a task entry or a regular operation in AdaTask and AdaTaskType classes. (Default = False)

IsExplicit

The boolean property IsExplicit allows you to specify that a constructor is an explicit constructor. (Default = False)

IsNative

The IsNative property specifies whether the Java modifier "native" should be added to an operation in the source file. The body of such operations, if specified, is ignored by the code generator.

Default = Cleared

JavaAnnotation

The property JavaAnnotation is used by the Rational Rhapsody code generator to insert Java annotations into generated code.

This property is used primarily for regenerating code that was reverse engineered. When you reverse engineer code that contains Java annotations, the value of the property JAVA_ReverseEngineering::ImplementationTrait::ImportJavaAnnotation determines how Rhapsody handles the annotation code. If the value of this property is set to Verbatim, then Rational Rhapsody does not import annotations as model elements. Rather, the annotation code is stored as the value of the property JavaAnnotation. When code is later regenerated, it will include the code that was stored in this property.

Default = Blank

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

Default = common

LocalVariablesDeclaration

The LocalVariablesDeclaration property specifies variables that you want to appear in the declaration of the entrypoint or operation. (Default = empty string)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

Me

The Me property specifies the name of the first argument to operations generated in C. (Default = me)

MeDeclType

The MeDeclType property is a string that specifies the type of the first argument to operations generated in C, as a pointer to an object or object type. The default value is as follows: $objectName* const The variable $objectName is replaced with the name of the object or object type.

PrivateQualifier

The PrivateQualifier property specifies the qualifier that is printed at the beginning of a private operation declaration or definition. You can set this property to an empty string to prevent the generation of the

static qualifier in the private function declaration or definition. (Default = static)

ProtectedName

The ProtectedName property specifies the pattern used to generate names of private operations in C. The default value is as follows: $opName The $opName variable specifies the name of the operation. For example, the generated name of a private operation go() of an object A is generated as: go()

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C. The default value is as follows: $objectName_$opName The $objectName variable specifies the name of the object; the $opName variable specifies the name of the operation. For example, the generated name of a public operation go() of an object A is generated as: A_go()

PublicQualifier

The PublicQualifier property specifies the qualifier that is printed at the beginning of a public operation declaration or definition. Note that the Static checkmark in the operation dialog UI is disabled in Rational Rhapsody Developer for C because the checkmark is associated with class-wide semantics that are not supported by Rational Rhapsody Developer for C. When loading models from previous versions, the Static check box is cleared; if the operation is public, the C_CG::Operation::PublicQualifier property value is set to Static in order to generate the same code. (Default = empty string)

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

RenamesKind

The RenamesKind property specifies whether the renaming of the operation designated in the Ada_CG::Operation::Renames property is “as specification” or “as body.”(Default = Specification)

ReturnTypeByAccess

The ReturnTypeByAccess property determines whether the return type is generated as an access type or a regular type. Note that this property is applicable only to classes for which an access type is generated. (Default = False)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you could add the @Deprecated annotation for an element by entering @Deprecated and a new line as the value of this property.

Default = Blank

TaskDefaultScheme

The TaskDefaultScheme property sets the task default entry scheme. The possible values are as follows:

- Conditional
- Timed
- None

(Default = None)

TaskDefaultSchemeDelayStatement

The TaskDefaultScheme property sets the task default entry statement for timed entry schemes. (Default = Empty MultiLine)

ThisByAccess

The ThisByAccess property specifies whether to pass the this parameter as an access mode parameter for a non-static operation. (Default = False)

ThisName

The ThisName property enables you to specify the name of the this parameter, which specifies the instance. (Default = this)

ThrowExceptions

The ThrowExceptions property specifies the exceptions that an operation can throw. Separate multiple exceptions with commas.

Default = Empty string

VirtualMethodGenerationScheme

The VirtualMethodGenerationScheme property enables backward-compatibility mode for methods of interface and abstract classes. The possible values are as follows:

- Default - The class type is class-wide, but the this parameters are not.
- ClassWideOperations - The class type is not class-wide, but the this parameters are.

(Default = Default)

Package

The Package metaclass contains properties that affect packages.

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

ContributesToNamespace

The ContributesToNamespace property specifies whether the packages contained in this package is declared as children packages of this package. Regardless of the setting, a directory is created for the current package to hold its contained elements. (Default = Checked)

DefineNameSpace

The DefineNameSpace property specifies whether a package defines a namespace. A namespace is a

declarative region that attaches an additional identifier to any names declared inside it.

Default = Checked

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

(Default = empty string)

EventsBaseID

The EventsBaseID property specifies the base ID for events. The default values are as follows:

- Default = 16

GenerateDirectory

The GenerateDirectory property specifies whether to generate a separate directory for the package.

The possible values are as follows:

- Checked - The package generates a directory. (This is the default.)
- Cleared - The package will not generate a directory.

Note that a directory is generated only if the GenerateDirectory and the DefineNameSpace properties are set to Checked.

ImpIncludes

The ImpIncludes property specifies the names (including full paths) of header files to be included at the top of implementation files generated for classes, objects or object types, or packages. Separate multiple file names using commas, without spaces. (Default = empty string)

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

ImplementationPragmasInContextClause

The ImplementationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the body. (Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.

- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

InitializationCode

The InitializationCode property specifies the user-defined initialization code to add to the package body. (Empty MultiLine)

IsNested

The IsNested property specifies whether to generate the class or package as nested. (Default = False)

IsPrivate

The IsPrivate property specifies whether to generate the class or package as private. (Default = False)

MarkPrologEpilogInAnnotations

The MarkPrologEpilogInAnnotations property specifies whether to generate ignore annotations for the Specification/Implementation Prolog/Epilog properties so they are ignored during roundtrip. When you insert code element declarations (variables, types, functions, and so on) in the Specification/Implementation Prolog/Epilog properties, after a full roundtrip those elements are added to the model and are duplicated on the next code generation. Using the MarkPrologEpilogInAnnotations property, you can have Rhapsody automatically ignore the information specified in the Specification/Implementation Prolog/Epilog properties instead of adding the ignore annotations manually. The possible values for the MarkPrologEpilogInAnnotations property are as follows:

- None - Rational Rhapsody does not generate any annotations. Any models created before Version 4.1 automatically have this property setting.
- Ignore - Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties.
- Auto - If the code in the Specification/Implementation Prolog/Epilog properties is one line (it does not contain any newline characters (\n)), no annotations are generated (the same behavior has the None setting). If there is more than one line, Rational Rhapsody generates the //#[ignore annotation before the code specified in the Specification/Implementation Prolog/Epilog properties, and generates the //#] annotation after the code specified in those properties (the same behavior as the Ignore setting).

During roundtrip, any ignore annotations in the comments of the element are not included in its description. Because the Specification/Implementation Prolog/Epilog properties are generated between the element's annotation and its declaration, you cannot rename those elements on roundtrip. If you change the name of an element, it is removed from the model and added with the new name. Some model information (for example, property settings) might be lost. (Default = Auto)

NestingVisibility

The NestingVisibility property specifies the visibility of the generated specification of the nested class or package. (Default = Public)

PackageClassNamePolicy

The PackageClassNamePolicy property specifies the naming policy for classes generated by Rhapsody. Rhapsody generates a class for each package in the Rational Rhapsody Developer for Java model. The possible values are as follows:

- Default - Use the default naming style (the package class name is the same as the package name).
- WithSuffix - Add a suffix to the class name. The suffix is "_pkgClass".

Default = WithSuffix

PackageEventIdRange

The PackageEventIdRange property specifies the maximum number of events allowed in a package. This property is set on the component level.

Default = 200

Renames

The Renames property enables one element to rename another element of the same type. You can also rename an element using a renames dependency. In the case of a conflict, the dependency has precedence. Note the following:

- For attributes, this property works only for static attributes in a class or for attributes in a package.
- For operations, this property contains the name of the operation this operation in renaming.The signatures of the two operations must match.

(Default = empty string)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationPragmas

The SpecificationPragmas property specifies the user-defined pragmas to generate in the specification. (Default = Empty MultiLine)

SpecificationPragmasInContextClause

The SpecificationPragmasInContextClause property specifies the user-defined pragmas to generate in the context clause of the specification. (Default = Empty MultiLine)

SpecificationProlog

The property SpecificationProlog allows you to add code to the beginning of the declaration of a model element.

For example, you could add the @Deprecated annotation for an element by entering @Deprecated and a new line as the value of this property.

Default = Blank

SpecIncludes

The SpecIncludes property specifies the names (including full paths) of header files to be included at the top of specification files generated for classes (C++ and Java), objects or object types (C), and packages. Separate multiple file names using commas, without spaces.

Default = Empty string

Port

The Port metaclass controls whether code is generated for ports.

Generate

The Generate property specifies whether to generate code for a particular type of element.

Default = Checked

Relation

The Relation metaclass contains properties that affect relations.

Add

The Add property specifies the command used to add an item to a container. (Default = Add_$target:c)

AddGenerate

The AddGenerate property specifies whether to generate an Add() operation for relations. (Default = Checked)

Animate

The Animate property specifies whether animation code is generated for an element. You can specify your own animation function using the property CG::Attribute::AnimSerializeOperation. The semantics of the Animate property is always in favor of the owner settings:

- If a package Animate property is set to False, all the classes owned by the package are not animated, regardless of the class Animate settings.
- If a class Animate property is set to False, all the elements in the class (attributes, operations, relations, and so on) are not animated.
- If an operation Animate property is set to False, all the arguments are not animated.
- If the AnimateArguments property is set to False, all the arguments are not animated, regardless of the specific argument Animate property settings.

(Default = Checked)

Clear

The Clear property specifies the name of an operation that removes all items from a relation. (Default = Clear_$target:c)

ClearGenerate

The ClearGenerate property specifies whether to generate a Clear() operation for relations. (Default = Checked)

CreateComponent

The CreateComponent property specifies the name of an operation that creates a new component in a composite class. (Default = New_$target:c)

CreateComponentGenerate

The CreateComponentGenerate property specifies whether to generate a CreateComponent operation for composite objects. Setting this property to False is one way to optimize your code for size. (Default = Checked)

DataMemberVisibility

The DataMemberVisibility property specifies the visibility of the relation data member. For example, if the relation is implemented as a pointer, this property determines whether the pointer data member is declared as public, private, or protected.

Default = Protected

DeleteComponent

The DeleteComponent property specifies the name of an operation that deletes a component from a composite class. (Default = Delete_$target:c)

DeleteComponentGenerate

The DeleteComponentGenerate property specifies whether to generate a DeleteComponent() operation for composite objects. (Default = Checked)

DescriptionTemplate

The DescriptionTemplate property specifies how to generate the element description in the code. An empty MultiLine (the default value) tells Rhapsody to use the default description generation rules. The property supports the following keywords:

- $Name - The element name
- $FullName - The full path of the element (P1::P2::C.a)
- $Description - The element description
- Element-specific keywords, as shown in the following table:

Metatype Describes Additional Supported Keywords Argument Arguments $Type - The argument type $Direction - The argument direction (in, out, and so on) Attribute Attributes $Type - The attribute type Class Classes, actors, objects, and blocks Event Events $Arguments - The event argument’s description Operation Primitive operations, triggered operations, $Arguments - The operation argument’s description constructors, and destructors $Signature - The operation signature Package Packages Relation Association ends $Target - The other end of the association Type Types $Type - Applicable to Typedef types

- Tag - The value of the specified element’s tag
- Property - The value of the element property with the specified name

The keywords are resolved in the following order:

- Predefined keywords (such as $Name)
- Tag keywords
- Property keywords

Note the following:

- Keyword names can be written in parentheses. For example: $(Name)
- If the value of a keyword is a MultiLine, each new line (except the first one) starts with the value of the lang_CG::Configuration::DescriptionBeginLine property; each line ends with the value of the property ADA_CG::Configuration::DescriptionEndLine.

(Default = empty string)

Find

The Find property specifies the name of an operation that locates an item among relational objects. (Default = Find_$target:c)

FindGenerate

The FindGenerate property specifies whether to generate a Find() operation for relations. (Default = False)

Get

The Get property specifies the name of an operation that retrieves the relation currently pointed to by the iterator. (Default = Get_$target:c)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index. The ContainerTypes>::Relationtype::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

Default = get$cname:cAt

GetAtGenerate

The GetAtGenerate property specifies whether to generate a getAt() operation for relations. The possible values are as follows:

- Checked - Generate a getAt() operation for relations.
- Cleared - Do not generate a getAt() operation for relations. Setting the GetAtGenerate property to False is one way to optimize your code for size.

Default = Cleared

GetEnd

The GetEnd property specifies the name of an operation that points the iterator to the last item in a collection. (Default = Get_$target:cEnd)

GetEndGenerate

The GetEndGenerate property specifies whether to generate a GetEnd() operation for relations. (Default = Checked)

GetGenerate

The GetGenerate property specifies whether to generate accessor operations for relations. (Default = Checked)

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key. For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

(Default = get$cname:c)

GetKeyGenerate

The GetKeyGenerate property specifies whether to generate getKey() operations for relations. Setting this property to Cleared is one way to optimize your code for size.

Default = Checked

ImplementationEpilog

The ImplementationEpilog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the end of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

ImplementationProlog

The ImplementationProlog property enables you to add any code that you want to be added as verbatim text (to be ignored by Rhapsody) to the beginning of the definition of a model element. For example, you could wrap a section of code with an #ifdef-#endif pair, add compiler-specific keywords, or add a #pragma statement. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

(Default = Empty MultiLine)

ImplementWithStaticArray

The ImplementWithStaticArray property specifies whether to implement relations as static arrays. The possible values are as follows:

- Default - Rational Rhapsody provides the appropriate implementation for all fixed and bounded relations.
- FixedAndBounded - All fixed and bounded relations are generated into static arrays.

To generate C-like code in C++ or Java, modify the value of the ImplementWithStaticArray property to FixedAndBounded.

Default = Default

InitializeComposition

The InitializeComposition property controls how a composition relation is initialized. The possible values are as follows:

- InInitializer
- InRecordType
- None

(Default = InInitializer)

Inline

The Inline property specifies how inline operations are generated. Which operations are affected by the Inline property depends on the metaclass:

- Attribute - Applies only to operations that handle attributes (such as accessors and mutators)
- Operation - Applies to all operations

- Relation - Applies only to operations that handle relations

Inlining in Rational Rhapsody Developer for Java Because inlining has no meaning in Java, the Inline property is set to none. (Default = none)

IsAliased

The IsAliased property is a Boolean value that specifies whether attributes are aliased. (Default = False)

JavaAnnotation

The property JavaAnnotation is used by the Rational Rhapsody code generator to insert Java annotations into generated code.

This property is used primarily for regenerating code that was reverse engineered. When you reverse engineer code that contains Java annotations, the value of the property JAVA_ReverseEngineering::ImplementationTrait::ImportJavaAnnotation determines how Rhapsody handles the annotation code. If the value of this property is set to Verbatim, then Rational Rhapsody does not import annotations as model elements. Rather, the annotation code is stored as the value of the property JavaAnnotation. When code is later regenerated, it will include the code that was stored in this property.

Default = Blank

Kind

The Kind property specifies the kind of operation that should be generated for an element. The kind of operations that can be generated is language-dependent (for example, virtual and abstract exist only in C++ and Java). In Java, Kind can be defined only for attributes and operations, but not for relations. This property affects class operations, in addition to accessors and mutators for relations and attributes. The possible values are as follows:

- common - Class operations and accessor/mutator are non-virtual.
- virtual - Class operations and accessor/mutator are virtual. This type is valid for C++ and Java only.
- abstract - Class operations and accessor/mutator are pure virtual. This type is valid for C++ and Java only.

(Default = common)

ObjectInitialization

The ObjectInitialization property specifies what kind of initialization will occur for the initial instances of a configuration. The possible values are as follows:

- Full - Instances are initialized and their behavior is started.
- Creation - Instances are initialized but their behavior is not started.
- None- Instances are not initialized and their behavior is not started.

(Default = Full)

Remove

The Remove property specifies the name of an operation that removes an item from a relation. (Default = Remove_$target:c)

RemoveGenerate

The RemoveGenerate property specifies whether to generate a Remove() operation for relations. (Default = Checked)

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = remove$cname:c

RemoveKeyGenerate

The RemoveKeyGenerate property specifies whether to generate a removeKey() operation for qualified relations. Setting this property to False is one way to optimize your code for size.

Default = Checked

RemoveKeyHelpersGenerate

The RemoveKeyHelpersGenerate property enables you to control the generation of the relation helper methods (for example, _removeItsX() and __removeItsX()). The possible values are as follows:

- True - Generate the helpers whenever code generation analysis determines that the methods are needed.
- False - Never generate the helpers.
- FromModifier - Generate the helpers based on the value of the CPP_CG::Relation::RemoveKey property.

Default = True

SafeInitScalar

The SafeInitScalar property specifies whether to initialize scalar relations as null pointers.

Default = Cleared

Set

The Set property specifies the name of the mutator generated for scalar relations. (Default = Set_$target:c)

SetGenerate

The SetGenerate property specifies whether to generate mutators for relations. (Default = Checked)

SpecificationEpilog

The property SpecificationEpilog allows you to add code to the end of the declaration of a model element.

Default = Blank

SpecificationProlog

The SpecificationProlog property enables you to add code to the beginning of the declaration of a model element (such as a configuration or class). For example, to create an abstract class in Java, you can set the SpecificationProlog property for the class to “abstract.” You must include the space after the word “abstract.” If the visibility for the class is set to default, the following class declaration is generated in the .java file: abstract class classname {...} The SpecificationProlog property allows you to add compiler-specific keywords, add a #pragma statement, or wrap a section of code with an #ifdef-#endif pair. For example, to specify that an operation is available only when the code is compiled with _DEBUG, set the following properties for the operation:

- Set SpecificationProlog to #ifdef _DEBUG cr.
- Set SpecificationEpilog to #endif.
- Set ImplementationProlog to #ifdef _DEBUG cr.
- Set ImplementationEpilog to #endif.

The following table lists whether leading and trailing linefeeds are generated.

Metaclass Leading Linefeed Added? Trailing Linefeed Added? Generated Inside or Outside or Namespace? Class Yes No Inside Package Yes Yes Inside

(Empty MultiLine)

Static

The Static property is a Boolean value that determines whether class-wide relations are enabled. Class-wide members of a class are shared between all instances of that class and are mapped as static. When a relation is tagged as static:

- The data member is generated as static (with the static keyword).
- The relation accessors are generated as static.

- The mutators of directional relations are generated as static. The mutators of symmetric relations are generated as common (non-static) operations.

Note the following behavior and restrictions:

- If there are links between instances based on static relations, code generation will initialize all the valid links. In case of a limited relation size, the last initialization is preserved.
- When you generate instrumented code (animation or tracing), relation NOTIFY calls are not added to static relation mutators.
- Animation associates static relations with the class instances, not the class itself.
- In an instrumented application (animation or tracing), the static relations names appear in each instance node; however, the values of directional static relations are not visible.

See also the properties CG::Relation::Containment, Containertype::Relationtype::CreateStatic, and Containertype::Relationtype::InitStatic.

Default = Cleared

Visibility

The Visibility property specifies the visibility of that kind of model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. (Default = Public)

Statechart

The Statechart metaclass contains the statechart code generation properties.

StatechartImplementation

Prior to version 7.3 of Rational Rhapsody, the transition-handling code generated by Rhapsody used a switch statement to represent the possible states. Beginning with version 7.3, this code uses an if/else structure. To allow older models to use the previous code generation behavior, a property called StatechartImplementation was added to the Pre73 backward compatibility profiles. The possible values for the property are:

- SwitchOnly - transition-handling code uses a switch statement to represent the possible states
- Default - the transition-handling code uses an if/else structure to represent the possible states

Default = SwitchOnly

Type

The Type metaclass contains a property that affects the visibility of data types.

AnimEnumerationTypeImage

The AnimEnumerationTypeImage property is a Boolean value that determines whether the Image attribute is used for enumerated types when using animation. (Default = False)

AnimSerializeOperation

The AnimSerializeOperation property enables you to specify the name of an external function used to animate all attributes and arguments that are of that type. Rhapsody can animate (display) the values of simple types and one-dimensional arrays without any problem. To display the current values of such attributes during an animation session, run the features window for the instance. However, if you want to animate a more complex type, such as a date, the type must be converted to a string (char *) for Rhapsody to display it. This is generally done by writing a global function, an instrumentation function , that takes one argument of the type you want to display, and returns a char *. You must disable animation of the instrumentation function itself (using the Animate and AnimateArguments properties for the function). For example, you can have a type tDate, defined as follows: typedef struct date { int day; int month; int year; } %s; You can have an object with an attribute count of type int, and an attribute date of type tDate. The object can have an initializer with the following body: me-date.month = 5; me-date.day = 12; me-date.year = 2000; If you want to animate the date attribute, the AnimSerializeOperation property for date must be set to the name of a function that will convert the type tDate to char *. For example, you can set the property to a function named showDate. This function name must be entered without any parentheses. It must take an attribute of type tDate and return a char *. The Animate and AnimateArguments properties for the showDate function must be set to False. The implementation of the showDate function might be as follows: showDate(tDate aDate) { char* buff; buff = (char*) malloc(sizeof(char) * 20); sprintf(buff,"%d %d %d", aDate.month,aDate.day,aDate.year); return buff; }

When you run this model with animation, instances of this object will display a value of 5 12 2000 for the date attribute in the browser. If the showDate function is defined in the same class that the attribute belongs to and the function is not static, the AnimSerializeOperation property value should be similar to the following:

myReal-showDate

This value shows that the function is called from the serializeAttributes function, located in the class OMAnimatedclassname. The showDate function must allocate memory for the returned string via the malloc/alloc/calloc function in C, or the new operator in C++. Otherwise, the system will crash. (Default = empty string)

AnimUnserializeOperation

The AnimUnserializeOperation property converts a string to the value of an element (the opposite of the AnimSerializeOperation property). Unserialize functions are used for event generation or operation invocation using the Animation toolbar to convert the string (received from the user) to the value of the event or operation before the event generation or operation invocation. For example, your serialization operation might look similar to the following:

char* myX2String(const Rec f) { char* cS = new char[OutputStringLength]; /* conversion from the Rec type to string */ return (cS); }

The unserialization operation would be: Rec myString2X (char* C, Rec T) { T = new Trc; /* conversion

of the string C to the Rec type */ delete C; return (T); }

(Default = empty string)

DeclarationPosition

The DeclarationPosition property specifies where the type declaration appears. The possible values are as follows:

- BeforeClassRecord - The type declaration appears before the class record (CR) declaration if CR has a visibility set to public, and before the class record forward declaration if CR has a visibility set to private.
- AfterClassRecord - The type declaration appears after the class record declaration if CR has a visibility set to public, and after the class record forward declaration if CR has a visibility set to private.
- StartOfDeclaration - The type declaration appears among the first declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the first declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.
- EndOfDeclaration - The type declaration appears among the last declarations (together with other types having the same settings) in the public section if CR has a visibility set to public, and among the last declarations in the private section (together with other types having the same settings) if CR has a visibility set to private.

(Default = BeforeClassRecord)

EnumerationAsTypedef

The EnumerationAsTypedef property specifies whether the generated enum should be wrapped by a typedef. This property is applicable to enumeration types in C and C++. (Default = Checked)

In

The In property specifies how code is generated when the type is used with an argument that has the modifier "In".

Default = $type

InOut

The InOut property specifies how code is generated when the type is used with an argument that has the modifier "InOut".

Default = $type

IsLimited

The IsLimited property determines whether the class or record type is generated as limited. (Default = False)

LanguageMap

The LanguageMap property specifies the Ada declaration for Rhapsody language-independent types. (Default = empty string)

Out

The Out property specifies how code is generated when the type is used with an argument that has the modifier "Out".

Default = $type

PrivateName

The PrivateName property specifies the pattern used to generate names of private operations in C. (Default = $typeName)

PublicName

The PublicName property specifies the pattern used to generate names of public operations in C. (Default = $objectName_$typeName)

ReferenceImplementationPattern

The ReferenceImplementationPattern property specifies how the Reference option for attribute/typedefs (composite types) is mapped to code. See the Rational Rhapsody Help for detailed information about using composite types. (Default = "")*

ReturnType

The ReturnType property specifies how code is generated when the type is used as a return type.

Default = $type

StructAsTypedef

The StructAsTypedef property specifies whether the generated struct should be wrapped by a typedef. This property is applicable to structure types in C and C++. (Default = Checked)

TriggerArgument

The TriggerArgument property is used for mapping event and triggered operation arguments to code instead of the In, InOut, and Out properties. A different property is required because of code generation limitations related to event arguments. See also:

- In
- InOut
- Out

Default = $type

UnionAsTypedef

The UnionAsTypedef property specifies whether the generated union should be wrapped by a typedef. This property is applicable to union types in C and C++. (Default = Checked)

Visibility

The Visibility property specifies the visibility of the model element. Code generation maps the visibility specified for an element to the same visibility in the generated language. The possible values are as follows:

- Public - The model element is public.
- Protected - The model element is protected.
- Private - The element is private.

(Default = Public)

JAVA_ReverseEngineering

In addition to the ReverseEngineering subject, Rational Rhapsody provides language-specific subjects to control how Rhapsody imports legacy code. Because most of the properties are identical for each language, they are represented with the JAVA tag, where JAVA can be C, CPP, or Java. Any language-specific properties are clearly labeled. In general, most of the reverse engineering (RE) properties have graphical representation in the Reverse Engineering Options window. You should change the options using this window instead of the corresponding properties. The metaclasses are as follows:

- Filtering
- ImplementationTrait
- Main
- Parser
- Promotions

Filtering

The Filtering metaclass contains properties that control which items are analyzed during the reverse engineering operation.

AnalyzeGlobalFunctions

The AnalyzeGlobalFunctions property specifies whether to analyze global functions. (Default = True)

AnalyzeGlobalTypes

The AnalyzeGlobalTypes property specifies whether to analyze global types. (Default = True)

AnalyzeGlobalVariables

The AnalyzeGlobalVariables property specifies whether to analyze global variables. (Default = True)

CreateReferenceClasses

The CreateReferenceClasses property specifies whether to create external classes for undefined classes that result from forward declarations and inheritance. By default, reference classes are created (as in previous versions of Rational Rhapsody). If the incomplete class cannot be resolved, the tool deletes the incomplete class if this property is set to Cleared. In some cases, the class cannot be deleted (for example, a class referenced by a typedef type).

Default = Checked

IncludeInheritanceInReference

The IncludeInheritanceInReference property specifies whether to include inheritance information in reference classes.

Default = Cleared

ReferenceClasses

The ReferenceClasses property specifies which classes to model as reference classes. Reference classes are classes that can be mentioned in the final design as placeholders without having to specify their internal details. For example, you can include the MFC classes as reference classes, without having to specify any of their members or relations. They would simply be modeled as terminals for context, to show that they are acting as superclasses or peers to other classes.

Default = empty string

ReferenceDirectories

The ReferenceDirectories property specifies which directories (and subdirectories) contain reference classes.

Default = empty string

ImplementationTrait

The ImplementationTrait metaclass contains properties that determine the implementation traits used during the reverse engineering operation.

AnalyzeIncludeFiles

The AnalyzeIncludeFiles property specifies which, if any, include files should be analyzed during reverse engineering. The possible values are as follows:

- AllIncludes—Analyze all include files.
- IgnoreIncludes—Ignore all include files.
- OnlyFromSelected—Analyze the specified include files only.
- OnlyLogicalHeader—Analyze the logical header files only.

Default = OnlyFromSelected

CreateDependencies

The CreateDependencies property (under C and JAVA_ReverseEngineering::ImplementationTrait) is used during reverse engineering (RE) for creating dependencies from include statements found in the imported code. This property determines whether the RE utility creates dependencies. Reverse engineering imports include statements as dependencies if the option Create Dependencies from Includes is set in the Rational Rhapsody GUI. This operation is successful if the reverse engineering utility analyzes both the included file and the source - and the source and included files contain class declarations for creating the dependencies between them. If there is not enough information, the includes are not converted dependencies. This can happen in the following cases:

- The include file was not found, or is not in the scope Input tab settings.
- A class is not defined in the include file or source file, so the dependency could not be created.

If the dependency is not created successfully, the include files that were not converted to dependencies are imported to the JAVA_CG::Class::SpecIncludes or ImpIncludes properties so you do not have to re-create them manually. If the include file is in the specification file, the information is imported to the SpecIncludes property; if it is in the implementation file, the information is imported to the ImpIncludes property. If a file contains several classes, include information is imported for all the classes in the file. The possible values for this property are as follows:

- None - Nothing is imported from include statements.
- DependenciesOnly - Model dependencies are created from include statements when it is possible to do so. This is the RE behavior of previous versions of Rational Rhapsody.
- All - The reverse engineering utility attempts to map the include file as a dependency. If it fails, the information is written to a property.

In previous versions of Rational Rhapsody, this property was a Boolean value. For backward compatibility, the old values are mapped as follows:

Old value Checked is mapped as new value DependenciesOnly

Old value Cleared is mapped as new value None

In addition to influencing reverse engineering, the CreateDependencies property also impacts the reverse engineering of user code added to model elements. The rules for interpreting #include and friend declarations for reverse engineering are as follows:

- Any #include OTHER in FILE is represented as a Uses dependency between each (outer) packages or classes in FILE to any (outer) packages or class in OTHER.
- If OTHER is not a specification file, the information is lost.
- If FILE is a specification file, the RefereeEffect is Specification. If FILE is an implementation file, the RefereeEffect is Implementation. Otherwise, the information is lost.

1. The way to decide if a file is a specification or an implementation file is defined elsewhere.

2. Any forward of a class or a package (via namespace) E in FILE is represented as a Uses dependency between each (outer) packages/classes in FILE to E. The RefereeEffect is Existence

3. This dependency is not added, if a Uses dependency can be matched.

4. Redundant Uses dependencies are removed. For example, when a relation is synthesized from a pointer to B, it is not necessary to add a Uses dependency.

5. A friend F (only when F is a class) of class C is is represented as a dependency with DependencyType to be Friendship from F to C.

Default = All

CreateFilesIn

The CreateFilesIn property is a placeholder for the reverse engineering option Create File-s In option. See the Rational Rhapsody Help for more information. You should not set this value directly. The default value for C is Package; the default values for the other languages is None.

DataTypesLibrary

The Mapping tab of the Reverse Engineering Options dialog allows you to specify a list of types that should be modeled as "Language" types. You can add individual types to the list or groups of types that you have previously defined as data types for a specific library.

If you select the option of adding a library, you are presented with a drop-down list of libraries to choose from. The libraries on this list are taken from the value of the property DataTypesLibrary. You can add a number of libraries to the drop-down list by using a comma-separated list of names as the value for this property.

When you select a library from the drop-down list, all of the types that were defined for that library are added to the list of types.

You define types for a library by carrying out the following steps:

- In the relevant .prp file, under the subject [lang]_ReverseEngineering, add a metaclass with the name of the library (using the same name you used in the value of the property DataTypesLibrary).
- Under the new metaclass, add a property called DataTypes.
- For the value of the DataTypes property that you added, enter a comma-separated list of the types that you want to include for that library.
- Now, if you select the library from the drop-down list displayed on the Mapping tab, the types you defined with the DataTypes property is automatically added to the list of types that should be modeled as "Language" types.

Default = Blank

ImportAsExternal

The property ImportAsExternal specifies whether the elements contained in the files you are reverse engineering should be brought into the model as "external" elements. This means that code will not be generated for these elements during code generation.

This property corresponds to the Import as External check box on the Mapping tab of the Reverse Engineering Options dialog.

Default = Cleared

ImportDefineAsType

The ImportDefineAsType property is a Boolean value that specifies how to import a #define. Note that models created before Version 5.2 automatically have this property overridden (set to True) when the model is loaded. The possible values are as follows:

- True—Import a #define as a user type.
- False—Import a #define as a constant variable, constant function, or type according to the following policy:
- If the #define has parameters, Rational Rhapsody creates a constant function. This applies to Rational Rhapsody Developer for C only.
- If the #define does not have parameters and its value includes only one line, Rational Rhapsody creates a constant variable. In Rational Rhapsody Developer for C++, the property CG::Attribute::ConstantVariableAsDefine is set to True.
- If the #define was not imported as a variable or function, Rational Rhapsody creates a type (the behavior of Rational Rhapsody 5.0.1).

Default = False

ImportJavaAnnotation

The property ImportJavaAnnotation allows you to specify how reverse engineering should handle Java annotations in your Java code. The property can take any of the following values:

- None - Code relating to Java annotations is ignored (and therefore annotations will not appear in the code that is later generated from the model).
- Model - All annotation-related code is brought into the model as elements that are visible in the browser (AnnotationType, JavaAnnotation, and AnnotationUsage).
- Verbatim - Code relating to Java annotations is processed. AnnotationTypes is brought into the model as visible elements, but annotation usage for individual elements will not be translated into elements in the model. For annotation usage, the text is stored using JavaAnnotation properties so that the annotations can be included in the code generated from the model.
- Mixed - Rational Rhapsody will try to bring annotation-related code into the model as visible elements. Where this is not possible, it will store the text as property values so that the annotations can be regenerated in the code.

Default = Verbatim

ImportStructAsClass

The ImportStructAsClass property is a Boolean value specifies how structs in external code are imported during reverse engineering. The possible values are as follows:

- True—structs are imported as classes (as in Rational Rhapsody 5.0 and earlier).
- False—structs are imported as types of kind Structure.

Default = False

MapToPackage

The property MapToPackage allows you to specify how the code elements you are reverse engineering should be divided into packages.

The property represents the options that appear in the Map to Package section of the Mapping tab in the Reverse Engineering Options dialog.

When the value of the property is set to Directory, a separate package is created for each subdirectory in the directory you have chosen to reverse engineer. The elements found in the files in each subdirectory is added to the package that corresponds to that subdirectory.

If you set the value of this property to User, then Rational Rhapsody will put all reverse engineered elements into a single package in the model. The name of the package is taken from the property [lang]_ReverseEngineering::ImplementationTrait::UserPackage.

Default = Directory

ModelStyle

The property ModelStyle determines how model elements are opened in the browser after reverse engineering - using a file-based functional approach or using an object-oriented approach based on classes (the corresponding property values are Functional and ObjectBased).

This property corresponds to the Modeling Policy radio buttons on the Mapping tab of the Reverse Engineering Options window.

Note that for C++ and Java, the file-based approach can only be used for visualization purposes. Rhapsody will not generate code from the model for elements imported using the Functional option. (You will notice that in the Reverse Engineering Options window, you can only select the File radio button if you first select the Visualization Only option.)

Default = Functional in RiC, ObjectBased in RiC++ and RiJ

PackageForExternals

If the value of the property UsePackageForExternals is set to True, the Rational Rhapsody reverse engineering feature puts all external elements in a separate package. You can control the name of this package by changing the value of the property PackageForExternals.

Default = Externals

PreCommentSensibility

During reverse engineering, a comment that comes immediately before the code for an element is considered a comment for that element, and the comment text is brought into Rational Rhapsody as the description for that element.

The property PreCommentSensibility is used to specify the maximum number of lines by which a comment can precede the code for an element and still be considered a comment for that element. Any comment that precedes an element by more than the number of lines specified is considered a global comment.

A value of 1 means that a comment must appear on the line prior to the code for an element to be considered a comment for that element.

Default = 2

ReflectDataMembers

The property ReflectDataMembers determines how the visibility of attributes is brought into the model when code is reverse engineered. The property affects both the visibility of the attribute in the regenerated code and the generation of get and set operations for the attribute. The property can take any of the following values:

- None - The visibility used for attributes is the same as that specified in the code that was reverse engineered. However, Rational Rhapsody generates public get/set operations for the attributes regardless of the visibility specified.
- VisibilityOnly - The visibility used for attributes is the same as that specified in the code that was reverse engineered. In addition, Rational Rhapsody generates get/set operations for the attribute with the same visibility. For example, if an attribute's visibility in the original code was private, the visibility is private in the regenerated code and the code will also include private get/set operations for the attribute.
- VisibilityAndHelpers - The visibility used for attributes is the same as that specified in the code that was reverse engineered. Rhapsody will not generate get/set operations for the attribute if the original code did not contain such operations.

Note that when the property is set to VisibilityAndHelpers, not only will get/set operations not be generated for attributes, but Rational Rhapsody does not generate any of its automatically-generated operations such as default constructors.

Default = VisibilityAndHelpers

RespectCodeLayout

The property RespectCodeLayout determines whether or not Rational Rhapsody saves information about the mapping of classes to files when reverse engineering code. The possible values for the property are:

- Mapping - Rational Rhapsody will remember which classes were contained in each of the files reverse engineered. When code is regenerated after reverse engineering, the classes is generated in the same files, such that if a file contained more than one class, it will still contain more than one class when the code is regenerated.
- None - Rational Rhapsody will not store any information regarding the mapping of classes to files. When code is regenerated after reverse engineering, each class is generated in its own file.

Default = None

RootDirectory

This property specifies the root directory for reverse engineering. This root directory may contain all the folders that should become package during the reverse engineering process. Rhapsody builds the package hierarchy according to the folder tree from the specified path.

Default = empty string

UseCalculatedRootDirectory

This property controls the use of the <lang>_ReverseEngineering::Implementation::RootDirectory property.

The possible values are:

- Never - Do not calculate the root directory.
- Always - Calculate the root directory and override the RootDirectory property.
- Auto - Ask the user if they want to override the value in the RootDirectory property if it is different from the calculated root directory. If the RootDirectory property is empty, Rational Rhapsody uses the calculated value without asking. This is the default value.

Default = Auto

UsePackageForExternals

When Rhapsody generates code, it does not regenerate code for elements that have been brought in as "external" elements. If you would like the reverse engineering feature to put all external elements into a separate package in the model, set the value of the property UsePackageForExternals to Checked. When a separate package is used, the name of the package is taken from the value of the property PackageForExternals.

Default = Cleared

UserDataTypes

The UserDataTypes specifies classes to be modeled as data types. This property corresponds to types entered in the Add Type window.

Default = empty string

UserPackage

When reverse engineering files, Rational Rhapsody allows you the option of having packages created for each subdirectory or having all of the reverse-engineered elements placed in a single package. This option is controlled by the property [lang]_ReverseEngineering::ImplementationTrait::MapToPackage.

When MapToPackage is set to "User", you can use the property UserPackage to provide the name that you would like Rhapsody to use for the single package that will contain all of the reverse-engineered elements.

You can specify a nested package by using the following syntax: package1::package2

If the model already contains a package with the specified name, the reverse-engineered elements are put in that package. If not, Rational Rhapsody will create the package.

This property corresponds to the text field provided for the package name in the Map to Package section of the Mapping tab in the Reverse Engineering Options dialog.

Default = ReverseEngineering

Main

The metaclass Main contains properties that define the file extensions used for filtering files in the reverse engineering file selection dialog, as well as properties that enable jumping to problematic lines of code by double-clicking messages in the Output window.

ErrorMessageTokensFormat

When errors are encountered during reverse engineering, they are displayed in the Rational Rhapsody Output window. If you double-click the error message, you are taken to the problematic line in the relevant source file.

This ability is made possible by the values provided for the properties ParseErrorMessage and ErrorMessageTokensFormat.

The value of the property ParseErrorMessage is a regular expression that extracts the relevant filename and line number information from the Rational Rhapsody -generated error message. The value of the property ErrorMessageTokensFormat is then used to interpret the information that was extracted from the error message.

The value of the property ErrorMessageTokensFormat consists of a comma-separated list of keyword-value pairs representing the number of tokens contained in the extracted information, which token represents the filename, and which token represents the line number.

Users should not change the value of this property.

Default = ToTalNumberOfTokens=2,FileTokenPosition=1,LineTokenPosition=2

ImplementationExtension

The property ImplementationExtension is only used for C and C++. It has no effect in Rational Rhapsody Developer for Java.

ParseErrorMessage

When errors are encountered during reverse engineering, they are displayed in the Rational Rhapsody Output window. If you double-click the error message, you are taken to the problematic line in the relevant source file.

This ability is made possible by the values provided for the properties ParseErrorMessage and ErrorMessageTokensFormat.

The value of the property ParseErrorMessage is a regular expression that extracts the relevant filename and line number information from the Rational Rhapsody -generated error message. The value of the property ErrorMessageTokensFormat is then used to interpret the information that was extracted from the error message.

Users should not change the value of this property.

Default = "([a-zA-Z_]+[:0-9a-zA-Z_.\/])"[:][ ]*LINE[ ]*([0-9]+)*

SpecificationExtension

The property SpecificationExtension is used to specify the filename extensions that should be used to filter files in the reverse engineering file selection dialog.

You can specify a number of extensions. They should be entered as a comma-separated list.

Default = java

MFC

The MFC metaclass contains a property that affects the MFC type library.

DataTypes

The DataTypes property specifies classes to be modeled as MFC data types. There is only one predefined library (MFC) that contains only one class (Cstring). You can, however, expand this short list of classes by the addition of classes in this property or the creation of new libraries in the property files factory.prpfactory and site.prpsite.

Default = Cstring

MSVC60

The MSVC60 metaclass contains properties used to control the Microsoft Visual C++ environment.

Defined

The Defined property specifies symbols that are defined for the Microsoft Visual C++ version 6.0 (MSVC60) preprocessor. These symbols are automatically filled into the Name list of the Preprocessing tab of the Reverse Engineering Options window when you select Add > Dialect: MSVC60. The default value is as follows:

__STDC__,__STDC_VERSION__,__cplusplus,__DATE__,
__TIME__,_WIN32,_cdecl,__cdecl,__int64=int,__stdcall,
__export,_export,_AFX_PORTABLE,_M_IX86=500,__declspec,
__MSC_VER=1200,__inline=inline,__far,__near,_far,_near,
__pascal,_pascal,__asm,__finally=catch,__based,
__inline=inline,__single_inheritance,__cdecl,__int8=int,
__stdcall,__declspec,__int16=int,__int32=int,__try=try,
__int64=int,__virtual_inheritance,__except=catch, __leave=catch,__fastcall,__multiple_inheritance)

IncludePath

The IncludePath property specifies necessary include paths for the Microsoft Visual C++ preprocessor. It is possible to specify the path to the site installation of the compiler as part of the site.prp, thus doing it only once and not for every project.

Default = empty string

Undefined

The Undefined property specifies symbols that must be undefined for the Microsoft Visual C++ preprocessor.

Default = empty string

Parser

The metaclass Parser contains properties that can be used to modify the way the parser handles code during reverse engineering.

Defined

The Defined property specifies symbols and macros to be defined using #define. For example, you can enter the following to define name> as text with the appropriate intermediate character: /D name{=|#}text

Default = empty string

Dialects

The Dialects property specifies which symbols are added to the Preprocessing tab of the Reverse Engineering window when that dialect is selected. The default value is MSVC60, which is itself defined by a metaclass of the same name under subject JAVA_ReverseEngineering. This dialect specifies the symbols that are defined for the Microsoft Visual C++ environment. You can define your own dialect (in the site.prp file) and select it in the Dialects property.

Default = Empty string.

IncludePath

The Preprocessing tab of the Reverse Engineering Options dialog allows you to specify an include path (classpath for Java) for the parser to use. The property IncludePath represents this path.

For the value of this property, you can enter a comma-separated list of directories. Note that you have to specify subdirectories individually.

The directories you list here is combined with the directories specified in #include statements in order to find the necessary files. For example, if you have c:\d1\d2\d3\file.h, you can enter c:\d1\d2 as the value of this property and then use d3\file.h in the #include statement.

You should take into account that the value of this property also determines the structure of the source file directory when code is generated from the model. So, in the above example, the top-level directory created is d3.

Default = Blank

Undefined

The Undefined property specifies symbols and macros to be undefined using #undef.

Default = empty string

Promotions

The metaclass Promotion contains a number of properties used to specify whether Rational Rhapsody should add various advanced modeling constructs to your model based on relationships/patterns uncovered during reverse engineering.

EnableAttributeToRelation

The property EnableAttributeToRelation is used to specify whether Rational Rhapsody should add Associations to the model for attributes whose type is another class in the model.

For example, if you have two classes, A and B, and B contains an attribute of type A, Rational Rhapsody will add an Association to the model reflecting this relationship.

Default = Checked

EnableResolveIncompleteClasses

Sometimes, during reverse engineering, Rational Rhapsody is not able to find the base class for a given class. The property EnableResolveIncompleteClasses is used to specify that if Rhapsody finds a class with the same name as the base class in a different location, it should assume that this class is the missing base class.

Default = Checked

JAVA_Roundtrip

The JAVA_Roundtrip subject contains properties that affect roundtripping. Most of the properties are used by all three languages. However, any language-specific properties are clearly labeled. The metaclasses are as follows:

- General
- Update

General

The General metaclass contains properties that control how changes to code are roundtripped in Rational Rhapsody.

NotifyOnInvalidatedModel

The NotifyOnInvalidatedModel property is a Boolean value that determines whether a warning window is displayed during roundtrip. This warning is displayed when information might get lost because the model was changed between the last code generation and the roundtrip operation. (Default = True)

ParserErrors

The ParserErrors property specifies the behavior of roundtrip when a parser error is encountered. The possible values are as follows:

- Abort - Abort roundtrip whenever there is a parser error in the code. No changes is applied to the model.
- AskUser - When Rhapsody encounters an error, it asks what you want to do.
- AbortOnCritical - Abort roundtrip if any critical parser errors are encountered in the code.
- Ignore - Continue roundtrip, ignoring any parser errors that are encountered.

Default = AskUser

PredefineIncludes

The PredefineIncludes property specifies the predefined include path for roundtripping.

Default = $OMROOT\LangJava\src,D:\jdk1.2.2\src

PredefineMacros

The PredefineMacros property specifies the predefined macros for roundtripping. The default value is as follows:

DECLARE_META(class_0\,animClass_0), DECLARE_REACTIVE_META(class_0\,animClass_0), OMINIT_SUPERCLASS(class_0Super\,animClass_0Super), OMREGISTER_CLASS\,DECLARE_META_T(class_0\, ttype\,animClass_0), DECLARE_REACTIVE_META_T(class_0\, ttype\,animClass_0), DECLARE_META_SUBCLASS_T(class_0\, ttype\,animClass_0), DECLARE_REACTIVE_META_SUBCLASS_T(class_0\, ttype\,animClass_0), DECLARE_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM), IMPLEMENT_META(class_0\,Default\,FALSE), IMPLEMENT_META_S(class_0\,FALSE\,class_1\,animClass_1\, animClass_0), IMPLEMENT_META_M(class_0\, FALSE\, class_0Super\, 2\,animClass_0), IMPLEMENT_REACTIVE_META(class_0\,Default\,FALSE), IMPLEMENT_REACTIVE_META_S(class_0\,FALSE\,class_1\, animClass_1\,animClass_0), IMPLEMENT_REACTIVE_META_M(class_0\, FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_REACTIVE_META_SIMPLE(class_0\,Default\,FALSE), IMPLEMENT_REACTIVE_META_S_SIMPLE(class_0\,FALSE\,class_1\ ,animClass_1\,animClass_0), IMPLEMENT_REACTIVE_META_M_SIMPLE(class_0\, FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_META_T(class_0\, Default\, FALSE\, animClass_0), IMPLEMENT_META_S_T(class_0\,FALSE\,class_0Super\,animclas s_0Super\,animClass_0), IMPLEMENT_META_M_T(class_0\, FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_META_OBJECT(class_0\,class_type\,Default\, FALSE), IMPLEMENT_META_S_OBJECT(class_0\,class_type\,FALSE\, class_1\,animClass_1\,animClass_0), IMPLEMENT_META_M_OBJECT(class_0\,class_type\,FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_REACTIVE_META_OBJECT(class_0\,class_type\, Default\,FALSE), IMPLEMENT_REACTIVE_META_S_OBJECT(class_0\,class_type\, FALSE\,class_1\,animClass_1\,animClass_0), IMPLEMENT_REACTIVE_META_M_OBJECT(class_0\,class_type\, FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_REACTIVE_META_SIMPLE_OBJECT(class_0\, class_type\,Default\,FALSE), IMPLEMENT_REACTIVE_META_S_SIMPLE_OBJECT(class_0\, class_type\,FALSE\,class_1\,animClass_1\,animClass_0), IMPLEMENT_REACTIVE_META_M_SIMPLE_OBJECT(class_0\, class_type\,FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_META_T_OBJECT(class_0\,class_type\, Default\, FALSE\, animClass_0), IMPLEMENT_META_S_T_OBJECT(class_0\,class_type\,FALSE\, class_0Super\,animclass_0Super\,animClass_0), IMPLEMENT_META_M_T_OBJECT(class_0\,class_type\, FALSE\, class_0Super\, 2 \,animClass_0), IMPLEMENT_MEMORY_ALLOCATOR(CLASSNAME\,INITNUM\, INCREMENTNUM\,ISPROTECTED), DECLARE_META_PACKAGE(Default), DECLARE_PACKAGE(Default), IMPLEMENT_META_PACKAGE(Default\,Default), DECLARE_META_EVENT(event_0), DECLARE_META_SUBEVENT(event_0\,event_0Super\, event_0SuperNamespace), IMPLEMENT_META_EVENT(event_0\,Default\,event_0), IMPLEMENT_META_EVENT_S(words\, words\, baseWords), DECLARE_OPERATION_CLASS(mangledName), DECLARE_META_OP(mangledName), OM_OP_UNSER(type\, name), OP_UNSER(func\, name), OP_SET_RET_VAL(retVal), OM_OP_SET_RET_VAL(retVal), IMPLEMENT_META_OP(animatedClassName\, mangledName\, opNameStr\, isStatic\, signatureStr\, numOfArgs), IMPLEMENT_OP_CALL(mangledName\, userClassName\, call\, retExp), STATIC_IMPLEMENT_OP_CALL(mangledName\, userClassName\, call\, retExp), OMDefaultThread=0, NULL=0, OMDECLARE_GUARDED OM_DECLARE_COMPOSITE_OFFSET

ReportChanges

The ReportChanges property defines which changes are reported (and displayed) by the roundtrip operation. The possible values are as follows:

- None - No changes are displayed in the output window.
- AddRemove - Only the elements added to, or removed from, the model are displayed in the output window.
- UpdateFailures - Only unsuccessful changes to the model are displayed in the output window.
- All - All changes to the model are displayed in the output window.

Default = AddRemove

RestrictedMode

The RestrictedMode property is a Boolean value that specifies whether restricted-mode roundtripping is enabled. This property can be modified on the configuration level. Restricted mode of full roundtrip enables you to roundtrip unusual usage of Rational Rhapsody elements, such as a class declaration in a user-defined type. Restricted mode has more limitations, but preserves the model from unexpected changes. The additional limitations for restricted mode are as follows:

- User-defined types cannot be removed or changed on roundtrip because Rational Rhapsody code generation adds the Ignore annotation for a user-defined type declaration.
- Relations cannot be removed or changed on roundtrip.
- New classes are not added to the model.

(Default = False)

RoundtripScheme

The RoundtripScheme property specifies whether to perform a basic or full roundtrip. Batch and online roundtrips change their behavior according to the specified value.

Default = Advanced

Type

The Type metaclass contains a property that controls whether user-defined types are ignored during the roundtrip operation.

Ignore

The Ignore property is a Boolean value that specifies whether to include user-defined types in a roundtrip operation. Types with the Ignore property set to True are generated with an Ignore annotation and will not be changed when a roundtrip is performed. The default value of this property is True, which allows no deletion or change to be done on types. Setting this property to False will reflect changes to the types declaration and deletion of types during roundtrip. Modifying the name of an existing type results in the addition of a new type, and removal of the model type (if the AcceptChanges property allows element removal), and the model's references to the removed type is lost (such as appearance in diagrams, property settings, and so on). You can set this property either on the configuration or on specific elements

in the model (which will affect itself and its aggregates). (Default = True)

Update

The Update metaclass contains a property that controls the update process used during roundtripping.

AcceptChanges

The AcceptChanges property is an enumerated type that specifies which changes are applied to each CG element (attribute, operation, type, class, or package). You can apply separate properties to each type of CG element. The possible values are as follows:

- All - All the changes can be applied to the model element.
- Default—1) Rhapsody will not roundtrip deletions if the updated code results in parser errors. 2) Rhapsody will not roundtrip the deletion of classes.
- NoDelete - All the changes except deletion can be applied to the model element. This setting prevents accidental removal of operations, constructors, attributes, relations, variables, instances, and functions.
- AddOnly - Apply only the addition of an aggregate to the model element. You cannot delete or change elements.
- NoChanges - Do not apply any changes to the model element.

Note that the value of the property is propagated to all the aggregates of an element. Therefore, if a package has the property value NoChanges, no elements in that package is changed.

Default = "Default"

UpdateExternalElements

Ordinarily, if an element in a model has been defined as an external element (meaning that the UseAsExternal property is set to Checked), Rational Rhapsody does not generate code for the element nor does it roundtrip into the model changes made to the element code.

However, if you set the value of the UpdateExternalElements property to Checked, Rational Rhapsody will roundtrip into the model changes made to the relevant external elements.

Default = Checked

Model

The Model subject contains properties that control prefixes added to attributes, variables, and arguments to reflect their type. The metaclasses are as follows:

- Attribute
- ControlledFile
- MatrixLayout
- MatrixView
- Profile
- Stereotype
- Type

Attribute

The Attribute metaclass contains a property that controls whether extra prefixes are added to attributes, variables, and arguments.

IsTemplateParameterType

Indicates that the attribute represents a template parameter. This property is used internally by Rhapsody. Under normal circumstances, there is no reason to modify the value of this property.

Prefix

The Prefix property specifies the prefix added to the model attributes, variables, and attributes of this type, if the property Model::Attribute::UseTypePrefix is set to Checked.

Note that when Rhapsody generates the code, the accessor and mutators do not include the prefix. For example, consider an attribute named A. If UsePrefix is set to Checked, Prefix is set to "m", and PrefixForAttribute is set to "t" and you change the attribute type:

- The accessor and mutator for the attribute is setA and getA.
- The actual name of the attribute is m_tA.

You can change the name of a variable, attribute, or argument so it does not obey the prefix. In this case, the element remains "unprefixed" until you change its type.

*Default = m_*

PrefixForStatic

The PrefixForStatic property specifies the extra prefix added to the model static attributes, if the property

Model::Attribute::UseTypePrefix is set to Checked.

Note that when Rhapsody generates the code, the accessor and mutators do not include the prefix. For example, consider an attribute named A. If UsePrefix is set to Checked, Prefix is set to "m", and PrefixForStatic is set to "s" and you change the attribute type:

- The accessor and mutator for the attribute is setA and getA.
- The actual name of the attribute is m_sA.

You can change the name of a variable, attribute, or argument so it does not obey the prefix. In this case, the element remains "unprefixed" until you change its type.

Note that template attributes do not use prefixes.

Default = s

UsePrefix

The UsePrefix property is a Boolean property that specifies whether prefixes are added to attributes, variables, and arguments to reflect their type. You set this property at the project level.

When this property is set to Checked, the name of the variable, attribute, or argument is updated automatically when you change the type of the variable, attribute, argument using the features window. However, the name is not changed automatically when the name of the type itself is changed.

Note the following restrictions:

- Template attributes do not use prefixes.
- Existing models are not automatically changed to obey the specified prefix. However, you can write a VBA macro to modify the model so it uses the prefixes.

Note that you specify the prefix added to the name by setting the properties Prefix, PrefixForAttribute and PrefixForStatic.

Default = Cleared

Class

Contains property that indicates whether the class represents a template parameter.

IsTemplateParameterType

Indicates that the class represents a template parameter. This property is used internally by Rhapsody. Under normal circumstances, there is no reason to modify the value of this property.

ControlledFile

The ControlledFile metaclass allows you to create controlled files and then use their features.

- Controlled Files, such as project specifications files (e.g. Word, Excel files) are typically added to a project for reference purposes and can be controlled through Rhapsody.
- A controlled file can be a file of any type (.doc, .txt, .xls, etc.).
- Controlled files are added into the project from the Rational Rhapsody browser.
- Controlled files can be added to diagrams via drag-and-drop from the browser.
- Currently, only Tag and Dependency features can be added to a controlled file.
- By default all controlled files are opened by their Windows-default programs (for example, Microsoft Excel for .xls files).
- The program(s) associated with controlled files can be changed via the Properties tab in the controlled files window.

DeleteUnderlyingFileWhenDeletingTheElement

The property DeleteUnderlyingFileWhenDeletingTheElement specifies whether Rational Rhapsody should delete the underlying file when a controlled file element is removed from a model. The possible values are:

- Never - the underlying file should not be deleted.
- Always - the underlying file should be deleted.
- AskUser - the user should be asked whether the underlying file should be deleted when the element is removed from the model.

Default = AskUser

FileTypes

The property FileTypes can be used to filter the files shown in the file browsing dialog that is displayed when the user creates a new controlled file and the property Model::ControlledFile::NewPolicy is set to the value Browser.

*Default = *.**

NewCommand

The property NewCommand specifies the command that should be executed when the user selects the option of adding a new controlled file. This command is executed only if the property Model::ControlledFile::NewPolicy is set to the value UseNewCommand.

Default for UNIX = touch $name.ext

Default for Windows = $OMROOT\etc\touch.exe "$file.ext"

NewPolicy

The property NewPolicy determines how controlled files are created. The possible values are:

- Browse - when you right-click on an object in the browser and select Add NewControlled File, a standard file browsing window is displayed. The dialog will display all file types, unless you have modified the property Model::ControlledFile::FileTypes.
- UseNewCommand - when Add NewControlled File is selected from the browser, the command in the property NewCommand is executed. The default value of the command is "touch $name.ext" for UNIX and "$OMROOT\etc\touch.exe "$file.ext"" for Windows. The "$file" is expanded to a file name for the controlled file. If a stereotype is specified, Rational Rhapsody uses the stereotype as the file name and adds a sequence of numbers to it. If no stereotype is specified, then it uses the name 'Controlled_File' and adds a sequence number to the end of the file name.

Default = Browse

OpenCommand

The property OpenCommand specifies the command that should be executed when the user opens a controlled file from the Browser. This command is executed only if the property Model::ControlledFile::OpenPolicy is set to the value UseOpenCommand.

Default = $fileName

OpenFileAfterCreation

If the boolean property OpenFileAfterCreation is set to Checked, Rational Rhapsody opens controlled files immediately after they are created.

Default= Cleared

OpenPolicy

The property OpenPolicy determines how Rhapsody opens controlled files. The possible values are:

- SystemDefault - opens the file using the MIME-type mapping for the system.
- UseOpenCommand - uses the command specified in the property Model::ControlledFile::OpenCommand.
- UseRhapsodyCodeEditor - opens the file using the Rational Rhapsody internal code editor.
- UseRhapsodyCSVFileViewer - opens the file using the Rational Rhapsody internal CSV file viewer.
- UseRhapsodyTableViewer - opens the file using the Rational Rhapsody table viewer. This option is only relevant for CSV files.
- AskUser - dialog is opened, allowing the user to select one of the open methods.

Default = SystemDefault

WaitTimeAfterFileCreation

The property WaitTimeAfterFileCreation determines the amount of time (in seconds) that Rational Rhapsody waits for an external command to be executed to create a new controlled file before control is returned to Rational Rhapsody.

Default = 1

MatrixLayout

The MatrixLayout metaclass contains properties that you can use for the design of matrix layouts.

ShowContainerElementForPorts

This property instructs Rhapsody to look at Ports as well as its container elements when displaying From/To information of Links and Flows.

Default = Checked

MatrixView

The MatrixView metaclass contains properties that you can use for the appearance of matrix views.

HideCellNames

This property show or hides Rhapsody element names in matrix view cells. Select this option if you want to hide element names (an icon appears instead to indicate the element type).

Default = Cleared

HideEmptyRowsCols

This property shows or hides rows and columns that are not holding any element data. This property also reflects the last selected mode set in the MatrixView toolbar.

Default = Cleared

Profile

The Profile metaclass contains a property that specifies the behavior of profiles.

AdditionalHelpersFiles

The property AdditionalHelpersFiles can be used to specify additional .hep files that should be associated with a profile, beyond the .hep file associated with the profile because it shares the same name as the profile.

The value of this property should be a comma-separated list of the additional .hep files you would like to associate with the profile.

Default = Blank

AnimateSDLBlockBehavior

By default, in animated sequence diagrams, SDLBlocks are considered to be black boxes. Internal events within the block are not displayed. If you would like the animated diagrams to display these internal SDL events, set the value of this property to True.

This property is set at the profile level.

Default = Cleared

PropertyFile

The property PropertyFile allows you to specify an additional .prp file that should be associated with the profile. Enter the path to the relevant .prp file.

Note that in terms of priority, if the file specified here has a property with the same name as a property specified at the factory, site, or profile level, the possible values and default value in this file will take precedence.

Default = Blank

SDLSignalPrefix

The naming convention used for the Rational Rhapsody events that represent SDL signals is to add "_" as a prefix to the original signal name. The property SDLSignalPrefix allows you to change this prefix.

This property is set at the profile level.

Default = _

UseRapidPorts

By default, SDLBlocks use behavioral ports. The property UseRapidPorts can be used to change this behavior. When set to True, rapid ports is used instead.

This property is set at the profile level.

Default = Cleared

Stereotype

The Stereotype metaclass includes properties that relate to the use of stereotypes in general, and to the use of "new term" stereotypes in particular.

Aggregates

When you create a "new term" stereotype, the property Aggregates is used to specify what types of elements can be added to this type of element. This is the list of elements that is included in the "Add New" context menu for elements of this type.

If the value of this property consists of more than one element, the element names should be separated by commas.

If the property is left blank, then the aggregates of the base element are used.

Default = Blank

AllowedTypes

Ordinarily, when you create an object, you can select the class on which it should be based from all of the available classes in the model. However, when you define a "new term" stereotype that is applicable to Objects, you can use the property AllowedTypes to limit the classes on which such objects can be based.

When you create an object of this "new term" type, Rational Rhapsody will only allow you to base it on one of the classes that is listed in the value of the property AllowedTypes for the relevant "new term" stereotype.

Default = Blank

AlternativeDrawingTool

In certain cases, a number of different out-of-the-box drawing elements are based on the same metaclass, for example, both Class and Composite Class are based on a metaclass called Class. So, when you add a custom diagram element using a "new term" stereotype, then in addition to specifying the base metaclass

in the Applicable to: field, you have to provide the name of the desired base element in the value of the property AlternativeDrawingTool.

This property does not have to be used if you are basing your new element on the "default" element of the metaclass.

This information is used for situations where the ambiguity has to be removed, such as determining what icon is used to represent the "new term" on a drawing toolbar, if the user has not specified a custom icon.

Default = Blank

BrowserGroupIcon

When you define a "new term" stereotype, you can use the property BrowserGroupIcon to specify an icon that should be used in the browser to represent this category of elements.

Provide the full path to the icon file (.ico).

When entering the path, the extension ".ico" is optional.

Default = Blank

BrowserIcon

When you define a "new term" stereotype, you can use the property BrowserIcon to specify an icon that should be used in the browser to represent individual elements of that type.

Provide the full path to the icon file (.ico).

When entering the path, the extension ".ico" is optional.

Default = Blank

CommentNotation

The CommentNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (under Comment:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and includes an ability to add compartments to that box.

Default = Note_Style

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the MakeDefault option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ConstraintNotation

The ConstraintNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Constraint:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

CustomHelpBookName

The property CustomHelpBookName is used in conjunction with the properties CustomHelpMapFile and CustomHelpURL to provide Rhapsody with the necessary parameters for displaying profile-specific context-sensitive help if you have prepared such help text for your profile.

Default = Blank

CustomHelpMapFile

The property CustomHelpMapFile is used in conjunction with the properties CustomHelpBookName and CustomHelpURL to provide Rhapsody with the necessary parameters for displaying profile-specific context-sensitive help if you have prepared such help text for your profile.

The value of this property should be the URL of the map file, for example, \\share\dodaf_help\dodaf_help.map. You can include environment variables in the URL, for example, $DODAF_HLP_ROOT\dodaf_help.map.

Default = Blank

CustomHelpURL

The property CustomHelpURL is used in conjunction with the properties CustomHelpBookName and CustomHelpMapFile to provide Rhapsody with the necessary parameters for displaying profile-specific context-sensitive help if you have prepared such help text for your profile.

The value of this property should be the URL of the help file, for example, \\share\dodaf_help\main_dodaf_help.html. You can include environment variables in the URL, for example, $DODAF_HLP_ROOT\main_dodaf_help.html.

Default = Blank

DrawingShape

When you define a "new term", you can use the property DrawingShape to customize the way the element will appear when added to a diagram. This property can take any of the following values:

- Default - the appearance of the "new term" is the same as that of the element on which it is based
- BasicBox - the "new term" element will appear as a rectangular box
- RoundedBox - the "new term" element will appear as a rectangular box with rounded edges

Default = "Default"

DrawingToolbar

If you have defined a custom diagram using a "new term" stereotype, you can use the property DrawingToolbar to provide a comma-separated list of elements that should be included in the drawing toolbar for that type of diagram, for example, RpyDefault,RpySeparator,Firewire. (RpyDefault represents all the elements included in the drawing toolbar of the base diagram.)

The list can include elements supported by the base diagram, and any "new terms" based on these elements.

The order of appearance in the toolbar will reflect the order specified in this property.

If the value is left blank, the tools from the base diagram is opened.

Default = Blank

DrawingToolIcon

When you define a "new term" stereotype to create a new type of diagram or diagram element, you can use the property DrawingToolIcon to specify a custom icon that should be used to represent the diagram/diagram element in toolbars.

If you are adding a new type of diagram, this is the icon that appears in the Rational Rhapsody "Diagram" toolbar.

If you are adding a new type of diagram element, this is the icon that appears in the drawing toolbar for the diagram on which it appears. (Keep in mind that the icon will only appear in the toolbar if the corresponding element is included in the list of elements specified for the property DrawingToolbar for that diagram.)

If you leave the value of this property blank, Rational Rhapsody uses the icon that you specified for display in the browser, using the property BrowserIcon. If the value of BrowserIcon was also left blank, the icon for the base element is used.

Default = Blank

DrawingToolTip

When you define a "new term" that will appear in a drawing toolbar, you can use the DrawingToolTip property to provide a tooltip that is opened when you mouse over the icon that was defined for the new element using the DrawingToolIcon property (for both diagrams and diagram elements).

Default = Blank

HideTabsInFeaturesDialog

When you create new types of model elements using the "new term" stereotype mechanism, the Features dialog for such elements will contain the tabs that are included in the Features dialog for the element on which the "new term" is based. The property HideTabsInFeaturesDialog gives you the option to hide some of these tabs.

To use this feature, enter for the value of this property a comma-separated list of the tabs that you would like to hide, for example:

Description,Relations,Tags

Default = Blank

InitialLayoutForTables

This property binds a (table or matrix) layout and view through the use of a New Term stereotype that is applicable to table or matrix view elements. If a stereotype is applicable to a table or matrix view, you can set the InitialLayoutForTables property with an existing table or matrix layout name to bind a table or matrix view to its corresponding layout.

Note that such a stereotype must reside within a profile for this property to work.

Default = Blank

Name

When you define a "new term", the property Name can be used to specify the name that should be used to represent this type of element in the "Add New" context menu.

If the value of the property is left blank, Rational Rhapsody uses the name that was given to the "new term" stereotype in the Features dialog.

Default = Blank

Owners

When you create a "new term" stereotype, the property Owners is used to specify the types of elements to which this type of element can be added.

If the value of this property consists of more than one element, the element names should be separated by commas.

If the value of the property is left blank, then the owners of the base element are used.

Default = Blank

PluralName

When you define a "new term", the property PluralName can be used to specify the string that should be used to represent the plural of this type of element in the browser.

If the value of the property is left blank, Rational Rhapsody will just add an "s" to the name used for individual elements of this type.

Default = Blank

PropertyFile

When you create a "new term" stereotype, you can use the property PropertyFile to specify a property file (.prp) that should be added to factory.prp for any project that contains this stereotype.

Default = Blank

RequirementNotation

The RequirementNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Requirement:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available options:

- Name
- Description
- Label

ShowAttributes

The boolean property ShowAttributes determines whether attributes is opened. When a stereotype is applied to an element that contains attributes, such as a class, the value of this property will override the value of ShowAttributes at the element level. This property will not override the element-level property if the user has overridden the element-level property.

Default = Explicit

ShowOperations

The boolean property ShowOperations determines whether operations is opened. When a stereotype is applied to an element that contains operations, such as a class, the value of this property will override the value of ShowOperations at the element level. This property will not override the element-level property if the user has overridden the element-level property.

Default = Explicit

TableLayout

The TableLayout metaclass contains properties that you can use for the design of the table layout.

ShowContainerElementForPorts

This property instructs Rhapsody to look at Ports as well as its container elements when displaying

From/To information of Links and Flows.

Default = Checked

Type

The Type metaclass contains properties that specify which extra prefixes are added to attributes, variables, and arguments for a specific type.

PrefixForAttribute

The PrefixForAttribute property specifies the extra prefix added to the model attributes, variables, and attributes of this type, if the property Model::Attribute::UseTypePrefix is set to True.

Note that when Rhapsody generates the code, the accessor and mutators do not include the prefix. For example, consider an attribute named A. If UsePrefix is set to Checked, Prefix is set to "m", and PrefixForAttribute is set to "t" and you change the attribute type:

- The accessor and mutator for the attribute is setA and getA.
- The actual name of the attribute is m_tA.

You can change the name of a variable, attribute, or argument so it does not obey the prefix. In this case, the element remains “unprefixed” until you change its type.

Default = Empty string

ObjectModelGe

The ObjectModelGe properties determine the appearance and behavior of object model diagrams. The metaclasses are as follows:

- Actor
- Aggregation
- Association
- Attribute
- AutoPopulate
- Class
- ClassDiagram
- Comment
- Complete
- Composition
- Constraint
- ContainArrow
- Depends
- Flow
- Inheritance
- Link
- Note
- Object
- Package
- PrimitiveOperation
- Requirement
- Type
- UseCase

Actor

The Actor metaclass contains properties that control the appearance of actors in object model diagrams.

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Aggregation

The Aggregation metaclass contains a property that controls the appearance of aggregation lines in object model diagrams.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowSourceMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "source" end of the line.

Default = Checked

ShowSourceQualifier

The boolean property ShowSourceQualifier is used for a number of connector elements, such as Associations. When set to True, the source element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_0. The property can be set at the diagram level. The default value for this property is True for object model diagrams and False for use case diagrams. The Display Options dialog can be used to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

Default = Checked

ShowSourceRole

The boolean property ShowSourceRole is used for a variety of connector elements, such as Links and Associations. When set to True, the source end of the relationship is displayed alongside the connector, for example itsClass_1. The property is set at the diagram level.

Default = Cleared

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

ShowTargetMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "target" end of the line.

Default = Checked

ShowTargetQualifier

The boolean property ShowTargetQualifier is used for a number of connector elements, such as Associations. When set to Checked, the target element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_1. The property can be set at the diagram level.

Default = Checked

You can use the Display Options window to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

ShowTargetRole

The boolean property ShowTargetRole is used for a variety of connector elements, such as Links and Associations. When set to True, the target end of the relationship is displayed alongside the connector, for example itsClass_2. The property is set at the diagram level.

Default = Cleared

Association

The Association metaclass contains properties that control the appearance of association lines in object model diagrams.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowSourceMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "source" end of the line.

Default = Checked

ShowSourceQualifier

The boolean property ShowSourceQualifier is used for a number of connector elements, such as Associations. When set to True, the source element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_0. The property can be set at the diagram level. The default value for this property is True for object model diagrams and False for use case diagrams. The Display Options dialog can be used to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

Default = Checked

ShowSourceRole

The boolean property ShowSourceRole is used for a variety of connector elements, such as Links and Associations. When set to True, the source end of the relationship is displayed alongside the connector, for example itsClass_1. The property is set at the diagram level.

Default = Cleared

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

ShowTargetMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "target" end of the line.

Default = Checked

ShowTargetQualifier

The boolean property ShowTargetQualifier is used for a number of connector elements, such as Associations. When set to Checked, the target element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_1. The property can be set at the diagram level.

Default = Checked

You can use the Display Options window to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

ShowTargetRole

The boolean property ShowTargetRole is used for a variety of connector elements, such as Links and Associations. When set to True, the target end of the relationship is displayed alongside the connector, for example itsClass_2. The property is set at the diagram level.

Default = Cleared

Attribute

The Attribute metaclass contains properties that control attributes in object model diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0

- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name_only

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Bottom-Top

LayoutStyle

The LayoutStyle property is used when Rhapsody automatically generates a diagram, and it determines the general appearance of the diagram - hierarchical or orthogonal.

- Hierarchical - diagram layout will reflect a hierarchy, appropriate for relationships such as inheritance.
- Orthogonal - diagram layout will resemble a grid, appropriate where there are no clear hierarchical relationships between the elements in the diagram.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Hierarchical

Class

The Class metaclass contains properties that control the appearance of new class boxes drawn in object model diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowAttributes

The ShowAttributes property specifies which attributes are shown in an object box in a component diagram. The possible values are as follows:

- All - Show all attributes.
- None - Do not show any attributes.
- Public - Show only the public attributes.
- Explicit - Show only those attributes that you have explicitly selected.

Default = Explicit

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Relative

ShowOperations

The ShowOperations property specifies which operations to show in an object box in a component or

object model diagram. The possible values are as follows:

- All - Show all operations.
- None - Do not show any operations.
- Public - Show only the public operations.
- Explicit - Show only those operations that you have explicitly selected.

Default = Explicit

ShowPorts

The ShowPorts property is a Boolean value that determines whether ports are displayed in object model diagrams.

Default = Checked

ShowPortsInterfaces

The ShowPortsInterfaces property is a Boolean value that determines whether port interfaces are displayed in object model diagrams.

Default = Checked

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

ClassDiagram

The ClassDiagram metaclass contains a property that controls the default fill color of class diagrams in object model diagrams.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

TreeContainmentStyle

Rhapsody allows you to display namespace containment in object model diagrams. This type of notation is also referred to as “alternative membership notation.” It depicts the hierarchical relationship between elements and the element that contains them, for example:

- requirements that contain other requirements
- packages that contain classes
- classes that contain other classes

The ability to display namespace containment is controlled by the boolean property TreeContainmentStyle, which can be set at the diagram, package, or project level. Namespace containment can only be displayed if the property is set to True.

Default = False (Note that in the SysML profile the default value of the property is True.)

If you have enabled the display of namespace containment by setting the value of the property to True, you can then display namespace containment as follows:

Drag the “container” element and the “contained” elements to the diagram. Then, from the menu, select Layout > Complete Relations > All.

The hierarchical relationship between the elements are depicted in the diagram.

Alternatively, you can select the Populate Diagram option when creating a new diagram. If you then select elements that have a hierarchical relationship, the diagram created will display the namespace containment for the elements.

Note that there is no drawing tool to manually draw this type of relationship on the canvas. Containment relationships between elements can only be displayed automatically based on existing relationships, using one of the methods described above.

Comment

The Comment metaclass contains properties that control comments in object model diagrams.

CommentNotation

The CommentNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (under Comment:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and includes an ability to add compartments to that box.

Default = Note_Style

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available options:

- Name
- Description
- Label

Default = Description

ShowForm

Determines how note-like elements are opened. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows

you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

Composition

The Composition metaclass contains a property that controls the appearance of compositions in object model diagrams.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

RepresentParts

The property RepresentParts determines what type of element is added to the Rational Rhapsody browser when you draw a composition connector (black diamond) in an object model diagram.

By default, when you add a composition relationship to a diagram, you will see an Association added in the browser.

If you prefer that Rational Rhapsody display this relationship as a Part in the browser, set the value of this property to True.

Default = Cleared

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element

- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowSourceMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "source" end of the line.

Default = Checked

ShowSourceQualifier

The boolean property ShowSourceQualifier is used for a number of connector elements, such as Associations. When set to True, the source element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_0. The property can be set at the diagram level. The default value for this property is True for object model diagrams and False for use case diagrams. The Display Options dialog can be used to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

Default = Checked

ShowSourceRole

The boolean property ShowSourceRole is used for a variety of connector elements, such as Links and Associations. When set to True, the source end of the relationship is displayed alongside the connector, for example itsClass_1. The property is set at the diagram level.

Default = Cleared

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

ShowTargetMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "target" end of the line.

Default = Checked

ShowTargetQualifier

The boolean property ShowTargetQualifier is used for a number of connector elements, such as Associations. When set to Checked, the target element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_1. The property can be set at the diagram level.

Default = Checked

You can use the Display Options window to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

ShowTargetRole

The boolean property ShowTargetRole is used for a variety of connector elements, such as Links and Associations. When set to True, the target end of the relationship is displayed alongside the connector, for example itsClass_2. The property is set at the diagram level.

Default = Cleared

Constraint

The Constraint metaclass contains properties that control the constraints in object model diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ConstraintNotation

The ConstraintNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Constraint:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available options:

- Name
- Description
- Label

Default = Description

ShowForm

Determines how note-like elements are opened. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

ContainArrow

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

Depends

The Depends metaclass contains a property that controls the appearance of dependency relation lines in object model diagrams.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various

constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = None

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Flow

The Flow metaclass contains properties that control how information flows are displayed in object model diagrams.

flowKeyword

The flowKeyword property is a Boolean value that specifies whether the flow keyword for the information flow is displayed in the diagram.

Default = Checked

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

ShowConveyed

The property ShowConveyed determines whether or not flow items should be displayed alongside the flows that convey them, and if so, what text should be displayed for the flow items. The property can take any of the following values:

- Name - the name of the flow item
- Label - the label of the flow item
- None - nothing should be displayed for the flow item

Note that this property only affects the display of new flows added to a diagram. The display of flow items for flows already on a diagram can be controlled by selecting the Display Options... item from the context menu for flows.

Default = Name

Inheritance

The Inheritance metaclass contains a property that controls the appearance of inheritance lines in object model diagrams.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = None

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Link

The Link metaclass contains a property that controls how links are displayed in object model diagrams.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations,

depending on the starting and ending points of the line.

- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowSourceMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "source" end of the line.

Default = Cleared

ShowSourceQualifier

The boolean property ShowSourceQualifier is used for a number of connector elements, such as Associations. When set to True, the source element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_0. The property can be set at the diagram level. The default value for this property is True for object model diagrams and False for use case diagrams. The Display Options dialog can be used to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

Default = Checked

ShowSourceRole

The boolean property ShowSourceRole is used for a variety of connector elements, such as Links and Associations. When set to True, the source end of the relationship is displayed alongside the connector, for example itsClass_1. The property is set at the diagram level.

Default = Cleared

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

ShowTargetMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "target" end of the line.

Default = Cleared

ShowTargetQualifier

The boolean property ShowTargetQualifier is used for a number of connector elements, such as Associations. When set to Checked, the target element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_1. The property can be set at the diagram level.

Default =

You can use the Display Options window to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

ShowTargetRole

The boolean property ShowTargetRole is used for a variety of connector elements, such as Links and Associations. When set to True, the target end of the relationship is displayed alongside the connector, for example itsClass_2. The property is set at the diagram level.

Default = Cleared

Note

The Note metaclass contains a property that controls how notes are displayed in object model diagrams.

ShowForm

Determines how note-like elements are opened. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

Object

The Object metaclass contains properties that control the appearance of objects drawn in object model diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties dialog or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

MultilineNameCompartment

The MultilineNameCompartment property specifies whether the name compartment for objects supports names that span more than one line.

Default = Cleared

ShowAttributes

The ShowAttributes property specifies which attributes are shown in an object box in a component diagram. The possible values are as follows:

- All - Show all attributes.
- None - Do not show any attributes.
- Public - Show only the public attributes.
- Explicit - Show only those attributes that you have explicitly selected.

Default = Public

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Relative

ShowOperations

The ShowOperations property specifies which operations to show in an object box in a component or object model diagram. The possible values are as follows:

- All - Show all operations.
- None - Do not show any operations.
- Public - Show only the public operations.
- Explicit - Show only those operations that you have explicitly selected.

Default = Public

ShowPorts

The ShowPorts property is a Boolean value that determines whether ports are displayed in object model diagrams.

Default = Checked

ShowPortsInterfaces

The ShowPortsInterfaces property (under ObjectModelGe::Class/Object) is a Boolean value that determines whether port interfaces are displayed in object model diagrams.

Default = Checked

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

ShowStereotypeOfClass

The property ShowStereotypeOfClass is a Boolean property that specifies whether or not the stereotypes of an object's class should be displayed on the object element when it is added to diagrams.

Default = Checked

Package

The Package metaclass contains properties that control the appearance of packages in object model diagrams.

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

PrimitiveOperation

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties dialog or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name_only

Requirement

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

RequirementNotation

The RequirementNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of two styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Requirement:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of three available options:

- Name
- Description
- Label

Default = Description

ShowForm

Determines how note-like elements are opened. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Stereotype

The Stereotype metaclass contains properties that relate to Stereotype elements in object model diagrams.

Compartments

The Compartments property allows you to specify which compartments should be shown when stereotype elements are displayed in object model diagrams. For the value of this property, you enter a comma-separated list of the compartments you would like Rhapsody to show. The list can include any of the elements that can be added to a stereotype.

Default = Tag

ShowName

The ShowName property allows you to specify how the element name should be shown when stereotype elements are displayed in object model diagrams.

The possible values are:

- Full_path - The full path of the element, beginning at the level of the root package
- Relative - The relative path of the element
- Name_only - The name of the element only
- Label - The label of the element only

Default = Name_only

Tag

The Tag metaclass contains properties that relate to Tag elements in object model diagrams.

Compartments

The Compartments property allows you to specify which compartments should be shown when tag elements are displayed in object model diagrams. For the value of this property, you enter a comma-separated list of the compartments you would like Rhapsody to show. The list can include any of the elements that can be added to a tag.

Default = Blank

ShowName

The ShowName property allows you to specify how the element name should be shown when tag elements are displayed in object model diagrams.

The possible values are:

- Full_path - The full path of the element, beginning at the level of the root package
- Relative - The relative path of the element
- Name_only - The name of the element only
- Label - The label of the element only

Default = Name_only

Type

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)

- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name_only

UseCase

The UseCase metaclass contains properties that control the appearance of use cases in object model diagrams.

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).

- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

OMContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- EmbeddedFixed - Defines properties for implementing embedded fixed relations.
- EmbeddedScalar - Defines properties for implementing embedded scalar (one-to-one) relations.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Contains properties that enable you to set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.
- You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.
- For example, you can change the definition of the Implementation property as follows:

Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->add($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target* The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target $cname() (Default = $CType $cname)*

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

(Default = new $CType)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = OMList<$RelationTargetType>)

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item) The default is $cname->find($item).

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is $cname->getAt($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:

$cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

(Default = <oxf/omlist.h>)

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator($cname).

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection:

*$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*(Default = *$iterator)*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator.reset().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $IterGetCurrent)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator

You can change the iterator type to one of your own choice. (Default = OMIterator<$RelationTargetType>)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname->remove($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

The default is $cname->removeAll().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item)) The default is $cname->add($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target* The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

The default is $CType $cname($multiplicity).

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType($multiplicity).

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = OMCollection<$RelationTargetType>)

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

The default is $cname->find($item).

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname->getAt($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection:

$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

(Default = <oxf/omcollec.h>)

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

The default is $cname($multiplicity).

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to

Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator($cname).

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection:

*$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*(Default = *$iterator)*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

(Default = $iterator++)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin The default is $iterator.reset().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $IterGetCurrent)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

(Default = OMIterator<$RelationTargetType>)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname->remove($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

The default is $cname->removeAll().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

EmbeddedFixed

The EmbeddedFixed metaclass defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

(Default = $CType)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = $(constant)$target $cname[$multiplicity])

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $RelationTargetType $cname[$multiplicity])

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = &$cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $RelationTargetType) &$cname[$index].

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

(Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

(Default = $IterType $iterator = 0;)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is (($RelationTargetType)&$cname[$iterator]).

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterIncrement)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterIncrement)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Blank

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default value is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $iterator < $multiplicity)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

(Default = int)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $(constant)$target$reference)

Remove

The Remove property specifies the command used to remove an item from a relation. For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers. For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

EmbeddedScalar

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $(constant)$target.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType$reference $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = &$cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library

convention for “finding” where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $(constRT)$target*.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $CType)*

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Fixed

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->add($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $CType $cname($multiplicity).

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to

Reference.

The default is new $CType($multiplicity).

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is OMCollection<$RelationTargetType>.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

The default is $cname->find($item).

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname->getAt($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection:

$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

(Default = <oxf/omcollec.h>)

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

The default is $cname($multiplicity).

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator($cname).

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection:

*$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*(Default = *$iterator)*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

The default is $iterator.reset().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $IterGetCurrent)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

(Default = OMIterator<$RelationTargetType>)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other

subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default value is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default value is $cname->remove($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

The default value is $cname->removeAll().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed:

$cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation

itself, because there is only one class involved:

$cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

General

The General metaclass contains properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

Qualified

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class

specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->add($keyName,$item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target*

(Default = $cname = new $CType)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

(Default = $CType $cname)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

(Default = new $CType)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = OMMap<$keyType, $RelationTargetType>)

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

The default is $cname->find($item).

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname->getAt($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection:

$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for “finding” where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

The default is $cname->getKey($keyName).

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

(Default = <oxf/ommap.h>)

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator($cname).

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection:

*$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*(Default = *$iterator)*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

The default is $iterator.reset().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator !=

$cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $IterGetCurrent)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

(Default = OMIterator<$RelationTargetType>)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $(constant)$target$reference)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(),

$cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname->remove($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

The default is $cname->removeAll().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

The default is $cname->remove($keyName).

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Scalar

The Scalar metaclass defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $(constant)$target$reference.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization

cases.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $(constRT)$target$reference.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL.

vector$target*::const_iterator You can change the iterator type to one of your own choice.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C

is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $CType)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed.

The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation,

passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

(Default = $cname = $item)

Type

The Type property specifies the type of the container as a pointer to the relation.

StaticArray

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is as follows: $Loop { if (!$cname[pos]) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

(Default = $CType)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = $RelationTargetType $cname[$multiplicity])

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

(Default = $cname[$index])

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection:

$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

The default is as follows: $Loop { $cname[pos] = NULL; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the

container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

(Default = $IterType $iterator = 0;)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

(Default = $cname[$iterator])

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

(Default = $iterator++)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterIncrement)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterIncrement)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

(Default = $iterator = 0)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is ($iterator < $multiplicity) && $cname[$iterator].

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

(Default = int)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property

body.

The default value for C++ is as follows: for (int pos = 0; pos < $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $(constant)$target$reference)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows: ($Loop { if ($cname[pos] == $item) { $cname[pos] = NULL; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified

relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed:

$cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

The default is $cname[$index] = $item.

Type

The Type property specifies the type of the container as a pointer to the relation.

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->add($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

(Default = $CType $cname)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

(Default = new $CType)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = OMList<$RelationTargetType>)

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

The default is $cname->find($item).

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname->getAt($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection:

$cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from

a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

(Default = <oxf/omlist.h>)

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

The default is $cname().

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator($cname).

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection:

*$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*(Default = *$iterator)*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

The default is $iterator.reset().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $IterGetCurrent)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You

can change the iterator type to one of your own choice.

(Default = OMIterator<$RelationTargetType>)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $(constant)$target$reference)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed.

The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname->remove($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname-clear()

The default is $cname->removeAll().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed:
$cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

The default is $cname[$index] = $item.

Type

The Type property specifies the type of the container as a pointer to the relation.

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->add($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target*

(Default = $cname = new $CType)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

(Default = $CType $cname)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

(Default = new $CType)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

(Default = OMCollection<$RelationTargetType>)

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

The default is $cname->find($item).

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

(Default = $CType $cname)

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

(Default = $cname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname->getAt($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

(Default = <oxf/omcollec.h>)

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator($cname).

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

(Default = $IterCreate)

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*(Default = *$iterator)*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

(Default = $iterator++)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

(Default = $IterReset)

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

(Default = $IterReset)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

(Default = $iterator.reset())

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

(Default = $IterGetCurrent)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

(Default = OMIterator<$RelationTargetType>)

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

(Default = $(constant)$target$reference)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the

collection. Finally, the item at that position is erased.

pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is $cname->remove($item).

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname-clear()

The default is $cname->removeAll().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

User

The User metaclass defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves.

To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to

Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL.

vector$target*::const_iterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

OMCorba2CorbaContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMCorba2CorbaContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Contains properties that enable you to set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows: Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = sequence<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = sequence<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedFixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default =

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default =

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default =

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default =

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default =

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedScalar

Defines properties for implementing embedded scalar (one-to-one) relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default =

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default =

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default =

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default =

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default =

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing relations of fixed size.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = sequence<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With

OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(),

$cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty MultiLine

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

Default = Empty string

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = sequence<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following

command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various

containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $RelationTargetType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default = Empty string

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = <$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname[$multiplicity]

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = sequence<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = sequence<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default =

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(target)Seq

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $target

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = Empty string

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

OMCpp2CorbaContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Contains properties that enable you to set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows: Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMCollection<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omcollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedFixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedScalar

Defines properties for implementing embedded scalar (one-to-one) relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing relations of fixed size.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMCollection<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript

operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omcollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty MultiLine

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

Default = Empty string

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($keyName,$item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMMap<$keyType, $RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following

command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default = $cname->getKey($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/ommap.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various

containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = $cname->remove($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $RelationTargetType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default = $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

$Loop { if (!$cname[pos]) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType $cname[$multiplicity]

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

$Loop { $cname[pos] = NULL; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname[$iterator]

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the

return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = ($iterator < $multiplicity) && $cname[$iterator]

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property

body.

Default for C++ = for (int pos = 0; pos < $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos < $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

$Loop { if ($cname[pos] == $item) { $cname[pos] = NULL; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty MultiLine

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

Default = $cname[$index] = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMCollection<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omcollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

*Default = *$iterator*

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $IterGetCurrent

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = OMIterator<$RelationTargetType>

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = Empty string

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

OMCppOfCorbaContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Contains properties that enable you to set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows: Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMCollection<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omcollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedFixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedScalar

Defines properties for implementing embedded scalar (one-to-one) relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = $(constant)$(FixedTarget)Seq;*

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing relations of fixed size.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMCollection<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript

operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omcollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty MultiLine

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

Default = Empty string

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($keyName,$item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMMap<$keyType, $RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following

command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default = $cname->getKey($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/ommap.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various

containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = $cname->remove($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $RelationTargetType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default = $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

$Loop { if (!$cname[pos]) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType $cname[$multiplicity]

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

$Loop { $cname[pos] = NULL; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = int $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the

return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property

body.

Default for C++ = for (int pos = 0; pos < $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos < $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

$Loop { if ($cname[pos] == $item) { $cname[pos] = NULL; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = Empty MultiLine

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

Default = $cname[$index] = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMList<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omlist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the

collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add($item)

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMCollection<$RelationTargetType>

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = $cname->find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omcollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = OMIterator<$RelationTargetType> $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

*Default = $(constant)$(FixedTarget)Seq**

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$(PoaPrefix)$(MappedTarget)

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove($item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter:

$cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector:

new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname:

vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = Empty string

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item:

$cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container at() operation to retrieve the item at the indexed position:

$cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = strong

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it:

$cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body.

For example:

pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item:

$iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end()

With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector<>, as defined in the STL.

vector<$target*>::const_iterator You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default =

for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to "to-many" (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased.

pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items:

$cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved:

$cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

OMUContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMUContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- EmbeddedFixed - Defines properties for implementing embedded fixed relations.
- EmbeddedScalar - Defines properties for implementing embedded scalar (one-to-one) relations.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Defines the properties that set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed equentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows:

Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add((void) $item)*

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMUList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $cname->find((void) $item)*

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the

item at the indexed position: $cname->at($index)

Default = ($RelationTargetType) $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omulist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)*$iterator

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

*Default = *$iterator*

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = OMUIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove((void) $item)*

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add((void) $item)*

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMUCollection

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $cname->find((void) $item)*

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = ($RelationTargetType) $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.

- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omucollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)*$iterator

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

*Default = *$iterator*

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = OMUIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove((void) $item)*

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedFixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty MultiLine

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $(constant)$target $cname[$multiplicity]

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $RelationTargetType $cname[$multiplicity]

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = &$cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = ($RelationTargetType) &$cname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript

operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty MultiLine

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = (($RelationTargetType)&$cname[$iterator])

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Blank

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $multiplicity

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second,

is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedScalar

Defines properties for implementing embedded scalar (one-to-one) relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $(constant)$target

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType$reference $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = &$cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = gen_ptr pos; $IterType $iterator =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

*Default = $(constRT)$target**

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default = $cname

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

*Default = $CType**

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing relations of fixed size.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add((void) $item)*

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = $cname = new $CType($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType $cname($multiplicity)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType($multiplicity)

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMUCollection

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $cname->find((void) $item)*

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the

item at the indexed position: $cname->at($index)

Default = ($RelationTargetType) $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omucollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $cname($multiplicity)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)*$iterator

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

*Default = *$iterator*

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = OMUIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove((void) $item)*

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property (under OMContainers::General) specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty MultiLine

ContainerIncludes

The ContainerIncludes property (under OMContainers::General) specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

Default = Empty string

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add((void*)$keyName,(void) $item)*

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMUMap

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $cname->find((void) $item)*

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps:
$cname-operator[]($keyName)

Default = ($RelationTargetType) $cname->getKey((void*)$keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omumap.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set

OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)*$iterator

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various

containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

*Default = *$iterator*

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = OMUIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation

implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove((void) $item)*

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = $cname->remove((void*)$keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $(constant)$target$reference

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation

implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $(constRT)$target$reference

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $CType

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default = $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $Loop { if (!$cname[pos]) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType $cname[$multiplicity]

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = $cname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The

property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Loop { $cname[pos] = NULL; }

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $cname[$iterator]

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = ($iterator < $multiplicity) && $cname[$iterator]

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $Loop { if ($cname[pos] == $item) { $cname[pos] = NULL; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname->clear()

Default = Empty MultiLine

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

Default = $cname[$index] = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add((void) $item)*

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMUList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $cname->find((void) $item)*

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = ($RelationTargetType) $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omulist.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)*$iterator

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

*Default = *$iterator*

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = OMUIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove((void) $item)*

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $cname->add((void) $item)*

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = $cname = new $CType

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = new $CType

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = OMUCollection

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $cname->find((void) $item)*

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = $cname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the

item at the indexed position: $cname->at($index)

Default = ($RelationTargetType) $cname->getAt($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/omucollec.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator($cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = ($RelationTargetType)*$iterator

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterReset

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator.reset()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = $IterType

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

*Default = *$iterator*

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = OMUIterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $cname->remove((void) $item)*

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $cname->removeAll()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it

the name stored in $cname: vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default = Empty string

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The

variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property (under most of the ContainerTypes::RelationType metaclasses) specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Empty string

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

PanelDiagram

The PanelDiagram subject contains properties that determine the appearance and behavior of panel diagram elements. It contains the following metaclasses:

- ButtonArray
- DigitalDisplay
- Gauge
- General
- Knob
- Led
- LevelIndicator
- MatrixDisplay
- Meter
- OnOffSwitch
- PushButton
- Slider
- TextBox

ButtonArray

The ButtonArray metaclass contains properties that determine the appearance and behavior of button array controls on panel diagrams.

ButtonFont

The ButtonFont property lets you select the font to use for the text on the face of a push button control.

To change the value of the property, click the "..." button in the box next to the property value to open the Font window. The value of the property affects both buttons already on the panel diagram and new buttons added to the diagram. (The display of buttons already on the panel diagram changes only after you refresh the diagram.)

Default = Arial 10 NoBold NoItalic

Direction

The Direction property determines whether the button array controls are used to input data, display data, or both. The possible values are:

- In - The button arrays are only used to input data for the attribute to which it is bound.

- Out - The button arrays are only used to display data for the attribute to which it is bound.
- InOut - The button arrays are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for button array elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the button array.
- BindedElement - The name of the attribute that is bound to the button array.
- Name - The name of the button array element.
- None - No text is displayed.

Default = Name

DigitalDisplay

The DigitalDisplay metaclass contains properties that determine the appearance and behavior of digital display controls on panel diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for digital display elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the digital display.
- BindedElement - The name of the attribute that is bound to the digital display.
- Name - The name of the digital display element.
- None - No text is displayed.

Default = Name

Gauge

The Gauge metaclass contains properties that determine the appearance and behavior of gauge controls on panel diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for gauge elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the gauge.
- BindedElement - The name of the attribute that is bound to the gauge.
- Name - The name of the gauge element.
- None - No text is displayed.

Default = Name

General

The General metaclass contains properties that apply to panel diagrams, in general. It does not apply to the specific controls that can be added to panel diagrams.

Fillcolor

The Fillcolor property determines the color used for the background of the panel diagram. Note the following:

- If applied at the diagram level, it changes the background color of that diagram.
- If applied at the package level, it is used as the background color for all new panel diagrams in the package and it will also change the background color of all existing panel diagrams in the package unless the property was set at the diagram level for a given diagram.
- Similarly, if applied at the project level, it is used as the background color for all new panel diagrams in the project, unless the property was set directly for individual packages. The selected color is used as the background color for all existing panel diagrams in the project unless the property was set at the package/diagram level for individual packages/diagrams.

Default = 192,192,192 (RGB values)

Knob

The Knob metaclass contains properties that determine the appearance and behavior of knob controls on panel diagrams.

Direction

The Direction property determines whether the knob controls are used to input data, display data, or both. The possible values are:

- In - The knobs are only used to input data for the attribute to which it is bound.

- Out - The knobs are only used to display data for the attribute to which it is bound.
- InOut - The knobs are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for knob elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the knob.
- BindedElement - The name of the attribute that is bound to the knob.
- Name - The name of the knob element.
- None - No text is displayed.

Default = Name

Led

The LED metaclass contains properties that determine the appearance and behavior of LED controls on panel diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for LED elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the LED.
- BindedElement - The name of the attribute that is bound to the LED.
- Name - The name of the LED element.
- None - No text is displayed.

Default = Name

LevelIndicator

The LevelIndicator metaclass contains properties that determine the appearance and behavior of level indicator controls on panel diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for level indicator elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the level indicator.
- Name - The name of the level indicator element.
- None - No text is displayed.

Default = Name

MatrixDisplay

The MatrixDisplay metaclass contains properties that determine the appearance and behavior of matrix display controls on panel diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for matrix display elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the matrix display.
- BindedElement - The name of the attribute that is bound to the matrix display.
- Name - The name of the matrix display element.
- None - No text is displayed.

Default = Name

Meter

The Meter metaclass contains properties that determine the appearance and behavior of meter controls on panel diagrams.

ShowName

The ShowName property determines whether or not a caption is displayed for meter elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the meter.
- BindedElement - The name of the attribute that is bound to the meter.
- Name - The name of the meter element.
- None - No text is displayed.

Default = Name

OnOffSwitch

The OnOffSwitch metaclass contains properties that determine the appearance and behavior of on/off switch controls on panel diagrams.

Direction

The Direction property determines whether the on/off switch controls are used to input data, display data, or both. The possible values are:

- In - The on/off switches are only used to input data for the attribute to which it is bound.
- Out - The on/off switches are only used to display data for the attribute to which it is bound.
- InOut - The on/off switches are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for on/off switch elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the on/off switch.
- BindedElement - The name of the attribute that is bound to the on/off switch.
- Name - The name of the on/off switch element.
- None - No text is displayed.

Default = Name

PushButton

The PushButton metaclass contains properties that determine the appearance and behavior of push button controls on panel diagrams.

ButtonFont

The ButtonFont property lets you select the font to use for the text on the face of a push button control.

To change the value of the property, click the "..." button in the box next to the property value to open the Font window. The value of the property affects both buttons already on the panel diagram and new buttons

added to the diagram. (The display of buttons already on the panel diagram changes only after you refresh the diagram.)

Default = Arial 10 NoBold NoItalic

ShowName

The ShowName property determines whether or not a caption is displayed for push button elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the push button.
- BindedElement - The name of the attribute that is bound to the push button.
- Name - The name of the push button element.
- None - No text is displayed.

Default = Name

Slider

The Slider metaclass contains properties that determine the appearance and behavior of slider controls on panel diagrams.

Direction

The Direction property determines whether slider controls are used to input data, display data, or both. The possible values are:

- In - The sliders are only used to input data for the attribute to which it is bound.
- Out - The sliders are only used to display data for the attribute to which it is bound.
- InOut - The sliders are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for slider elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the slider.
- BindedElement - The name of the attribute that is bound to the slider.
- Name - The name of the slider element.
- None - No text is displayed.

Default = Name

TextBox

The TextBox metaclass contains properties that determine the appearance and behavior of text box controls on panel diagrams.

Direction

The Direction property determines whether text box controls are used to input data, display data, or both. The possible values are:

- In - The text boxes are only used to input data for the attribute to which it is bound.
- Out - The text boxes are only used to display data for the attribute to which it is bound.
- InOut - The text boxes are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for text box elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the text box.
- BindedElement - The name of the attribute that is bound to the text box.
- Name - The name of the text box element.
- None - No text is displayed.

Default = Name

QoS

The QoS (Quality of Service) properties provide performance and timing information. It contains the following metaclasses:

- Class
- Operation
- Resource

For detailed information on Quality of Service properties, refer to the book "Doing Hard Time."

Class

The Class metaclass contains properties that control the periodicity of messages, the period and jitter times, minimum interarrival times, and so on.

AverageArrivalTime

The AverageArrivalTime property specifies the average time taken between message arrivals for an active class with an episodic (also referred to as "aperiodic") arrival pattern.

Default = 0

BlockingTime

The BlockingTime property specifies the maximum amount of time that an active class (or operation) can be prohibited from executing by a lower-priority action or task. Blocking can occur when a lower priority action or class locks (in a mutually exclusive way) a resource (such as a class) that is required by a higher priority action or task.

Default = 0

Deadline

The Deadline property specifies the maximum amount of time allowed for all activity by an active class resulting from a message or event reception. This property refers to the maximum amount of time allowable to complete the required response to an initiating action or event (not just handling the message per se, but also performing the required actions).

Default = 0

EstExecutionTime

The EstExecutionTime property specifies the estimated time taken by active class to act on or handle a received message. Early on, the actual time might not be known. This property holds the estimate.

Default = 0

ExecutionTime

The ExecutionTime property specifies the average time taken by an active class to act on or completely handle a received message. This property is redundant with the execution times of the operations involved in the execution of the behavior. By allowing this specification at the active class (thread) level, high level schedulability analysis can be performed. This is normally a measured value.

Default = 0

IsPeriodic

The IsPeriodic property specifies whether an active class can activated or initiated periodically. A message is said to have an "arrival pattern," which can be periodic or episodic. A task (or thread or active class) is said to have an activation pattern or to "be periodic." Active classes can be periodically initiated.

Default = Cleared

Jitter

The Jitter property specifies the largest interval of time variance between the message eception or resulting task activation by an active class. Periodic messages are characterized by a period with which the messages arrive, and by jitter, which is the variation around the period with which messages actually arrive. Jitter is normally modeled as a uniform random process but always totally within the jitter interval.

Default = 0

MinimumInterarrivalTime

The MinimumInterarrivalTime property specifies the minimum time that must occur between message arrivals for an active class with an episodic arrival pattern. Message arrivals can be episodic or periodic. An episodic arrival pattern is inherently unpredictable, but it can still be bounded. Episodic messages can have a minimum interarrival time, a minimum time that must occur between message arrivals.

Default = 0

Period

The Period property specifies the average amount of time between messages received by an active class. This property applies only to an active class that is, in fact, periodically activated by those messages. An active class can receive messages and then queue them for handling later when its thread has processing focus.

Default = 0

Operation

The Operation metaclass contains properties that control the estimated operation execution times, budgeted time, and blocking times.

BlockingTime

The BlockingTime property specifies the worst case time the task can be blocked from execution, in nanoseconds (ns).

Default = 0

Budget

The Budget property specifies the amount of time allocated to the worst case execution of an operation, in nanoseconds (ns).

Default = 0

EstExecutionTime

The EstExecutionTime property specifies the estimated worst case execution time, in nanoseconds (ns).

Default = 0

ExecutionTime

The ExecutionTime property specifies the worst case execution time, in nanoseconds (ns).

Default = 0

Resource

The Resource metaclass contains a property that sets the priority ceiling of resources.

PriorityCeiling

The PriorityCeiling property specifies the priority of the highest priority task that can lock the resource.

Default = 0

ReverseEngineering

The ReverseEngineering subject contains properties that affect how Rhapsody deals with legacy code. The metaclasses are as follows:

- Main
- Progress
- Update

The ReverseEngineering properties determine how Rhapsody imports legacy code. In addition to the language-independent properties specified in this subject, Rational Rhapsody also includes three language-specific subjects:

- C_ReverseEngineering
- CPP_ReverseEngineering
- JAVA_ReverseEngineering

Main

The Main metaclass contains properties that determine which legacy files are to be imported, and specify the legal format for license file names used in reverse engineering.

EnableProgressDialog

This property determines whether or not the Enable Progress window is opened.

Default = Cleared

ExcludeFilesMatching

Use this property to exclude particular files/folders from being reversed engineered. Values should be comma-separated wildcard expressions (for example: res*, dish*). Any files or folders that match any of these wildcard expressions is excluded from the list of files that are to be reverse engineered.

Default = Empty string

Files

The Files property specifies legacy files to be imported. You select files in the Open window (Tools > Reverse Engineering > Add).

Default = empty string

License

The License property specifies the location of the license for the parser used by the Reverse Engineering tool.

Default = empty string

ReAnalyzeFiles

The ReAnalyzeFiles property is a Boolean value that determines whether the file that was analyzed once by the reverse engineering tool is reanalyzed during the same RE session. This property is used to improve performance.

Default = Cleared

UseTreeViewByDefault

Use this property to set if the tree view should be used as the default Reverse Engineering user interface. If the value of this property is set to Checked, Rational Rhapsody will display the tree view when the Reverse Engineering tool is opened.

Note that the value of this property might be updated when you close the Reverse Engineering user interface. For example, if you are using the list view before you close the Reverse Engineering user interface, the value of this property will be set to Cleared. In addition, note that he change is set for the property at the active configuration level.

Default = Checked

Progress

The Progress metaclass contains properties that control how the progress of the reverse engineering operation is reported.

AnalyzedCodeConstruct

The AnalyzedCodeConstruct property specifies the analyzed constructs on which to report. The possible values are as follows:

- Class - Report only the analyzed classes.
- File - Report only the analyzed files.
- All - Report all analyzed constructs.

Default = Class

InformationApproximated

The InformationApproximated property specifies how to handle information that can only be approximated. The possible values are as follows:

- Ignore - Ignore this information.
- Report - Report the situation.
- Abort - Terminate importing.

Default = Report

InformationLost

The InformationLost property specifies how to handle information that Rational Rhapsody knows is lost. The possible values are as follows:

- Ignore - Ignore this information.
- Report - Report the situation.
- Abort - Terminate importing.

Default = Report

ModelUpdate

The ModelUpdate property specifies which constructs to add to the model. The possible values are as follows:

- All - Add all recognized constructs to the model.
- Class - Add only recognized classes to the model.

Default = Class

ModelUpdatingFailed

The ModelUpdatingFailed property specifies how to handle a failed import. The possible values are as follows:

- Ignore - Ignore this information.
- Report - Report the situation.
- Abort - Terminate importing.

Default = Report

OutputFile

The OutputFile property specifies the name of the log file to which status and error messages are written

during the import process. The Log and Process options in the Reverse Engineering Options window determine which conditions are reported. The same messages are simultaneously written to the output window and the log file.

Default = ReverseEngineering.log

OutputWindow

The OutputWindow property specifies which reverse engineering messages are written to the output window. This can help speed up performance. The possible values are as follows:

- None - No messages are written to the output window.
- File - The names of processed files are written to the output window.
- Error - Only errors are written to the output window.
- All - All messages are written to the output window.

Default = File

ParsingError

The ParsingError property specifies how to report progress errors. The possible values are as follows:

- Ignore - Ignore this information.
- Report - Report the situation.
- Abort - Terminate importing.

Default = Report

TimeStampPerFile

The boolean property TimeStampPerFile allows you to specify that during reverse engineering Rhapsody should include timestamps in the log files to indicate when each file was reverse engineered. If the value is set to Checked, timestamps is included.

Default = Cleared

Update

The Update metaclass contains a property that controls whether imported packages and classes are merged or overwritten.

CreateObjectModelDiagrams

When you reverse engineer code with Rational Rhapsody, you have the option of specifying that Rational Rhapsody should automatically generate object model diagrams based on the code imported. The Model Updating tab of the Reverse Engineering Options dialog contains a check box labeled "Populate Object Model Diagram" that can be selected to activate this option.

This feature is controlled by the property CreateObjectModelDiagrams.

Note that when you select this option in the Reverse Engineering Options dialog, it will modify the value of this property for the currently-active Configuration.

Default = Checked

Policy

The Policy property specifies whether imported packages/classes should overwrite or be merged with existing ones.

Default = Overwrite

RiCContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The RiCContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- EmbeddedFixed - Defines properties for implementing embedded fixed relations.
- EmbeddedScalar - Defines properties for implementing embedded scalar (one-to-one) relations.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Defines the properties that set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed equentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

For example, you can change the definition of the Implementation property as follows:

Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine.

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $(CType)_addHead(&($me$cname), $item)

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = mecname = $(CType)_Create()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType) $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = RiCList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = $(CType)_getAt(&($me$cname), $index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.

- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCList.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $(CType)_Init(&($me$cname));

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = gen_ptr pos; $IterType $iterator = $CastRT&($me$cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $(CType)_get($iterator, pos)

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $(CType)_next($iterator, &pos)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = $IterReset

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $(CType)_first($iterator, &pos)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = !$(CType)_isDone($iterator, pos)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

*Default = $(CType) **

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in

the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $(CType)_remove(&($me$cname), $item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $(CType)_removeAll(&($me$cname))

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $(CType)_add(&($me$cname), $item)

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = mecname = $(CType)_Create($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType) $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = RiCCollection

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = $(CType)_getAt(&($me$cname), $index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCCollection.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $(CType)_Init(&($me$cname), $multiplicity);

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = unsigned pos; $IterType $iterator = $CastRT&($me$cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $(CType)_get($iterator, pos)

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $(CType)_next($iterator, &pos)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = $IterReset

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $(CType)_first($iterator, &pos)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = !$(CType)_isDone($iterator, pos)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

*Default = $(CType) **

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $(CType)_remove(&($me$cname), $item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $(CType)_removeAll(&($me$cname))

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedFixed

Defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty MultiLine

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $(constant)$target $cname[$multiplicity]

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $RelationTargetType $cname[$multiplicity]

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = &(mecname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = &((mecname)[$index])

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCList.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor

body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default = $Loop { $target_ctor(&(($me$cname)[pos])); }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = &((mecname)[$iterator])

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Empty string

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the

container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = $iterator < $multiplicity

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty MultiLine

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $Loop { $target_Cleanup(&(($me$cname)[pos])); }

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

EmbeddedScalar

Defines properties for implementing embedded scalar (one-to-one) relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Cast

The Cast property specifies the target.

Default = ($target)*

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default =

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $(constant)$target

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType$(reference) $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = &(mecname)

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default = $target_ctor(&($me$(cname)))

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = gen_ptr pos; $IterType $iterator =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default = mecname

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

*Default = $CType**

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default = memcpy((void)&(mecname) ,(void*)$item, sizeof($target))*

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Fixed

Defines properties for implementing relations of fixed size.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $(CType)_add(&($me$cname), $item)

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = mecname = $(CType)_Create($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType) $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = RiCCollection

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the

item at the indexed position: $cname->at($index)

Default = $(CType)_getAt(&($me$cname), $index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCCollection.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $(CType)_Init(&($me$cname), $multiplicity);

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $(CType)_setFixedSize(&($me$cname), RiCTRUE)

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = unsigned pos; $IterType $iterator = $CastRT&($me$cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $(CType)_get($iterator, pos)

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $(CType)_next($iterator, &pos)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = $IterReset

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $(CType)_first($iterator, &pos)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = !$(CType)_isDone($iterator, pos)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

*Default = $(CType) **

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $(CType)_remove(&($me$cname), $item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $(CType)_removeAll(&($me$cname))

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

General

Defines properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property (under OMContainers::General) specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

Default = Empty MultiLine

ContainerIncludes

The ContainerIncludes property (under OMContainers::General) specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

Default = Empty string

Qualified

Defines properties for implementing qualified relations, which are accessed via a key.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $(CType)_add(&($me$cname), (gen_ptr)$keyName, $item)

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = mecname = $(CType)_Create(NULL)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType) $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = RiCMap

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = $(CType)_getAt(&($me$cname), $index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default = $(CType)_getKey(&($me$cname), (gen_ptr)$keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.

- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCMap.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $(CType)_Init(&($me$cname), NULL);

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = gen_ptr pos; $IterType $iterator = $CastRT&($me$cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $(CType)_get($iterator, pos)

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $(CType)_next($iterator, &pos)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = $IterReset

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $(CType)_first($iterator, &pos)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = !$(CType)_isDone($iterator, pos)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

*Default = $(CType) **

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in

the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $(CType)_remove(&($me$cname), $item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $(CType)_removeAll(&($me$cname))

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default = $(CType)_removeKey(&($me$cname), (gen_ptr)$keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

Scalar

Defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default =

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default =

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default =

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default =

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default =

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default =

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container’s end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default =

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default =

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default =

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default =

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default =

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default =

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default =

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default =

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default =

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default =

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default =

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default =

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default =

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default = mecname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

StaticArray

Defines properties for implementing static arrays.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $Loop { if (!$me$cname[pos]) { mecname[pos] = $item; break; } }

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default =

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = Empty MultiLine

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType)

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = $RelationTargetType$cname[$multiplicity]

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default =

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = mecname[$index]

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default =

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor

body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = $Loop { mecname[pos] = NULL; }

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = gen_ptr pos; $IterType $iterator = $IterType $iterator = 0;

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = mecname[$iterator]

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following

command moves the $iterator ahead one item: $iterator++

Default = $iterator++

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Empty string

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $iterator = 0

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the

container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = ($iterator < $multiplicity) && (($me$cname)[$iterator])

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = int

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to

be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $Loop { if ($me$cname[pos] == $item) { mecname[pos] = NULL; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $Loop { mecname[pos] = NULL; }

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

Default = $cname[$index] = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $(CType)_addHead(&($me$cname), $item)

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = mecname = $(CType)_Create()

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType) $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = RiCList

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = $(CType)_getAt(&($me$cname), $index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCList.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $(CType)_Init(&($me$cname));

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = gen_ptr pos; $IterType $iterator = $CastRT&($me$cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $(CType)_get($iterator, pos)

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $(CType)_next($iterator, &pos)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterReset

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = $IterReset

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $(CType)_first($iterator, &pos)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = !$(CType)_isDone($iterator, pos)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

*Default = $(CType) **

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $(CType)_remove(&($me$cname), $item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname->clear()

Default = $(CType)_removeAll(&($me$cname))

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

UnboundedUnordered

Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = $(CType)_addHead(&($me$cname), $item)

Cast

The Cast property specifies the target.

Default = ($target$reference)

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when generating MISRA compliant code.

Default = ($IterType)

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = $(CType)_Cleanup(&($me$cname))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = mecname = $(CType)_Create($multiplicity)

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = $(CType) $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = RiCCollection

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = $(CType)_find(&($me$cname), $item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = $CType $cname

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = mecname

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = $(CType)_getAt(&($me$cname), $index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation

implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = <oxf/RiCCollection.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = $(CType)_Init(&($me$cname), $multiplicity);

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = unsigned pos; $IterType $iterator = $CastRT&($me$cname)

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = $(CType)_get($iterator, pos)

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = $(CType)_next($iterator, &pos)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = $IterIncrement

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = $IterIncrement

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = $IterReset

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = $(CType)_first($iterator, &pos)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = !$(CType)_isDone($iterator, pos)

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

*Default = $(CType) **

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = $(constant)$target$reference

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = $(CType)_remove(&($me$cname), $item)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = $(CType)_removeAll(&($me$cname))

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

User

Defines properties for user-defined implementations of relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname->push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname->insert(map<$keyType,$target*>::value_type($keyName,$item))

Default = Empty string

Cast

The Cast property specifies the target.

Default =

CastRT

This property defines the return type casting in an iterator creation call. The user can empty it, as when

generating MISRA compliant code.

Default = Empty string

Cleanup

The Cleanup property contains a pattern for the appropriate container destructor call.

Default = Empty string

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector<$target*>

Default = Empty string

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector<$target*> $cname()

Default = Empty string

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

Default =

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector<$target*> collection type determines the type of the variable cl on the left side of the assignment:

vector<Client*>* cl = new vector<Client*>(2);

In this case, the class Client is the $target of vector operations.

Default = Empty string

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname->find($item)

Default = Empty string

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about composite types.

Default = Empty string

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname

The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

Default = Empty string

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname->at($index)

Default = Empty string

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end()

This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined

using the properties GetEnd and GetEndGenerate under CG::Relation.

Default =

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

Default =

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file.

Default = weak

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Default = Empty string

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

Default = Empty string

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos<$multiplicity; pos++; $cname[pos]=NULL

Default = Empty string

InitSimple

The InitSimple property contains a pattern for the appropriate container initialization call.

Default =

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

Default =

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector<$target*>::const_iterator $iterator; $iterator=$cname>begin()

Default = Empty string

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

Default = Empty string

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator

This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

Default = Empty string

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++

Default = Empty string

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

Default = Empty string

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

Default = Empty string

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Empty string

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname->begin()

Default = Empty string

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

Default = Empty string

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname->end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

Default = Empty string

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector<$target*>::const_iterator

You can change the iterator type to one of your own choice.

Default = Empty string

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

Default for C++ = for (int pos = 0; pos $multiplicity; ++pos)

Default for C = int pos; for (pos = 0; pos < $multiplicity; ++pos)

Default for Java = for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

Default =

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Default = Empty string

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector<$target*>::iterator pos=find($cname>begin(), $cname>end(),$item);$cname>erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair<$keyType,$target*> p; p.second=$item; map<$keyType,$target*>::iterator pos=find($cname->begin(), $cname->end(),p); $cname->erase(pos)

Default = Empty string

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname->clear()

Default = Empty string

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Default =

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Default =

Type

The Type property specifies the type of the container as a pointer to the relation.

Default =

RoseInterface

The RoseInterface properties determine how Rhapsody imports models from Rational Rose®. The subject contains a single metaclass: Import.

Import

The Import metaclass contains properties that specify whether to import statecharts, object model diagrams, and nested packages. These properties correspond to choices made in the Import Options window of the Rose Importer.

ImportAssociationWithNoName

When importing models from Rose, Rational Rhapsody imports associations only if they have a name in the Rose model.

If you would like Rhapsody to import associations even if they do not have a name in the Rose model, set the value of the property ImportAssociationWithNoName to True.

Default = Cleared

ImportODiagrams

The ImportODiagrams property specifies whether to import Rose class (object) diagrams.

(Default = Checked)

ImportStateCharts

The ImportStateCharts property specifies whether to import Rose statecharts.

(Default = Checked)

InsertRoseSkinProfile

When importing from Rose, Rational Rhapsody presents you with a check box labeled Use Rose Look-and-feel, which lets you to indicate whether or not the profile should be added. The initial value of this check box is taken from the property InsertRoseSkinProfile.

Default = True

Keep in mind that the RoseSkin profile will change the values of certain properties, for example, it sets the value of the IsSavedUnit property to False for classes, components, diagrams, and packages.

SkinFileName

The name of the .sbs file that Rational Rhapsody should use to determine the format and other settings for importing from Rose. The name of the relevant .sbs file provided with Rational Rhapsody.

(Default = RoseSkin)

UseNestingPackageNames

The UseNestingPackageNames property specifies whether to indicate package nesting using underscores between package names.

All nested packages are flattened on import. In other words, all nested packages are at the same level after being imported. If this option is enabled, a nested package A::B is imported as A_B. Otherwise, it is imported simply as B, and is at the same level as package A, its parent.

(Default = Checked)

RTInterface

The RTInterface properties determine how Rhapsody interacts with requirements traceability tools. The metaclasses are as follows:

- DOORS
- ExportOptions

DOORS

The DOORS metaclass contains properties that enable you to define the installation directory, Rational DOORS® project name, location of Rational DOORS license file, and whether to run Rational DOORS in batch mode.

CheckForLastModifyDiagrams

The CheckForLastModifyDiagrams property is a Boolean value that specifies whether to check when the diagrams were last modified.

(Default = Checked)

ExportReadOnlyUnits

The ExportReadOnlyUnits property is a Boolean value that specifies whether to export read-only units to Rational DOORS.

(Default = Checked)

InstallationDir

The InstallationDir property specifies the path to the Rational DOORS installation. This property is optional, because Rational Rhapsody reads this information from the Windows registry. (Default = empty string)

LinkModuleName

The LinkModuleName property specifies the name of the link's set module in Rational DOORS.

(Default = Rhapsody_links)

LmLicenseFile

The LmLicenseFile property specifies the location of your Rational DOORS license. This property is optional, because Rational Rhapsody reads this information from the Windows registry.

(Default = empty string)

ModuleNameFromProject

The ModuleNameFromProject property is a Boolean value that specifies whether to get the name of the module from the Rational Rhapsody project.

(Default = Cleared)

ProjectName

The ProjectName property specifies the name of the Rational DOORS project entered in the Rational DOORS Interface window. Exporting and checking of data are disabled until you enter a project name.

(Default = empty string)

RunInBatchMode

The RunInBatchMode property specifies whether to run Rational DOORS in batch mode rather than interactive mode. This property corresponds to the option of the same name in the Rational DOORS Interface window. Note that you must use interactive mode if you want to navigate to Rational DOORS from the Rational Rhapsody browser.

(Default = Cleared)

ExportOptions

The ExportOptions metaclass contains properties that determine which Rhapsody items are exported to the requirement traceability tool (DOORS) and how they are mapped to Rational DOORS formal modules. Most of these options are set in the Export Options window of the Rational DOORS interface.

ActivityDiagrams

The ActivityDiagrams property is a Boolean value that specifies whether to export activity diagrams to Rational DOORS.

(Default = Checked)

ActivityStates

The ActivityStates property is a Boolean value that specifies whether to export activity states to Rational DOORS.

(Default = Checked)

ActivityTransitions

The ActivityTransitions property is a Boolean value that specifies whether to export activity transitions to Rational DOORS.

(Default = Checked)

Actors

The Actors property is a Boolean value that specifies whether to export actors to Rational DOORS.

(Default = Checked)

Associations

The Associations property is a Boolean value that specifies whether to export associations to Rational DOORS.

(Default = Checked)

Attributes

The Attributes property is a Boolean value that specifies whether to export attributes to Rational DOORS.

(Default = Checked)

Classes

The Classes property is a Boolean value that specifies whether to export classes to Rational DOORS.

(Default = Checked)

CollaborationDiagrams

The CollaborationDiagrams property is a Boolean value that specifies whether to export collaboration diagrams to Rational DOORS.

(Default = Checked)

Comments

The Comments property is a Boolean value that specifies whether to export comments to Rational DOORS.

(Default = Checked)

ComponentDiagrams

The ComponentDiagrams property is a Boolean value that specifies whether to export component diagrams to Rational DOORS.

(Default = Checked)

Components

The Components property is a Boolean value that specifies whether to export components to Rational DOORS.

(Default = Checked)

Configurations

The Configurations property is a Boolean value that specifies whether to export configurations to Rational DOORS.

(Default = Checked)

Constraints

The Constraints property is a Boolean value that specifies whether to export constraints to Rational DOORS.

(Default = Checked)

ControlledFiles

The ControlledFiles property indicates whether or not external files, such as project specifications files produced in Word or Excel, are accepted as sources for requirements in the Rational Rhapsody project.

(Default = Checked)

CreateModulePerPackage

The CreateModulePerPackage property is a Boolean value that specifies whether to create a separate formal module in Rational DOORS to correspond to each package in the Rational Rhapsody project.

If this property is set to Checked, one Rational DOORS formal module is created for each package selected in the Rational Rhapsody browser tree in the Rational DOORS Interface dialog. Otherwise, a single formal module named RHAPSODY_MODULE is created in Rational DOORS to which all design elements are exported.

(Default = Checked)

Dependencies

The Dependencies property is a Boolean value that specifies whether to export dependencies to Rational DOORS.

(Default = Checked)

Events

The Events property is a Boolean value that specifies whether to export events to Rational DOORS.

(Default = Checked)

ExportAllScope

The ExportAllScope property is a Boolean value that specifies whether to export all elements to Rational DOORS. This property corresponds to the Export All check box in the Rational DOORS Interface window.

(Default = Checked)

ExportLabels

The ExportLabels property is a Boolean value that specifies whether to export labels to Rational DOORS.

(Default = Cleared)

ExportPictures

The ExportPictures property is a Boolean value that specifies whether to export pictures to Rational DOORS.

(Default = Cleared)

Files

The Files property is a Boolean value that specifies whether to export files to Rational DOORS.

(Default = Checked)

FlowItems

The FlowItems property is a Boolean value that specifies whether to export FlowItems to Rational DOORS.

(Default = Checked)

Flows

The Flows property is a Boolean value that specifies whether to export information flows to Rational DOORS.

(Default = Checked)

Folders

The Folders property is a Boolean value that specifies whether to export folders to Rational DOORS.

(Default = Checked)

GlobalFunctions

The GlobalFunctions property is a Boolean value that specifies whether to export global functions to Rational DOORS.

(Default = Checked)

GlobalInstances

The GlobalInstances property is a Boolean value that specifies whether to export global instances to Rational DOORS.

(Default = Checked)

GlobalVariables

The GlobalVariables property is a Boolean values that specifies whether to export global variables to Rational DOORS.

(Default = Checked)

HyperLinks

The HyperLinks property is a Boolean value that specifies whether to export hyperlinks to Rational DOORS.

(Default = Checked)

Links

The Links property is a Boolean value that specifies whether to export links to Rational DOORS.

(Default = Checked)

Nodes

The Nodes property is a Boolean value that specifies whether to export nodes to Rational DOORS.

(Default = Checked)

ObjectModelDiagrams

The ObjectModelDiagrams property is a Boolean value that specifies whether to export object model diagrams to Rational DOORS.

(Default = Checked)

Operations

The Operations property is a Boolean value that specifies whether to export operations to Rational DOORS.

(Default = Checked)

Packages

The Packages property is a Boolean value that specifies whether to export packages to Rational DOORS.

(Default = Checked)

Ports

The Ports property is a Boolean value that specifies whether to export ports to Rational DOORS.

(Default = Checked)

PurgeOnDelete

The PurgeOnDelete property is a Boolean value that specifies whether to use hard deletion in Rational DOORS. With hard delete, the element and its link is deleted from the Rational DOORS database. With soft delete, the element is marked as deleted, but remains in the database so it can be recovered; the link is deleted. Note the following:

- When there is an extra element in Rational DOORS that does not exist in Rational Rhapsody, the system asks whether you want to delete it.
- If you soft delete an element and later create an element with the same name, a new shadow element is created in Rational DOORS, and the old one is not used.
- If you switch from soft delete to hard delete, the soft-deleted elements remain in Rational DOORS.

(Default = Checked)

Relations

The Relations property is a Boolean value that specifies whether to export relations to Rational DOORS.

(Default = Checked)

Requirements

The Requirements property is a Boolean value that specifies whether to export requirements to Rational DOORS.

(Default = Checked)

ScopeToExport

The ScopeToExport property is a list of selected packages and diagrams that is exported to Rational DOORS if ExportAllScope is Cleared. (Default = empty string)

SequenceDiagrams

The SequenceDiagrams property is a Boolean value that specifies whether to export sequence diagrams to Rational DOORS.

(Default = Checked)

StateCharts

The StateCharts property is a Boolean value that specifies whether to export statecharts to Rational

DOORS.

(Default = Checked)

States

The States property is a Boolean value that specifies whether to export states to Rational DOORS.

(Default = Checked)

Stereotypes

The Stereotypes property is a Boolean value that specifies whether to export stereotypes to Rational DOORS.

(Default = Checked)

StructureDiagrams

The StructureDiagrams property is a Boolean value that specifies whether to export structure diagrams to Rational DOORS.

(Default = Checked)

Swimlanes

The Swimlanes property is a Boolean value that specifies whether to export swimlanes to Rational DOORS.

(Default = Checked)

Tags

The Tags property is a Boolean value that specifies whether to export tags to Rational DOORS.

(Default = Checked)

Transitions

The Transitions property is a Boolean value that specifies whether to export transitions to Rational DOORS.

(Default = Checked)

Types

The Types property is a Boolean value that specifies whether to export types to Rational DOORS.

(Default = Checked)

UseCaseDiagrams

The UseCaseDiagrams property is a Boolean value that specifies whether to export use case diagrams to Rational DOORS.

(Default = Checked)

UseCases

The UseCases property is a Boolean value that specifies whether to export use cases to Rational DOORS.

(Default = Checked)

SequenceDiagram

The SequenceDiagram subject contains properties that determine the appearance and behavior of sequence diagrams. It contains the following metaclasses:

- Condition_Mark
- General
- InstanceLine
- InteractionOperator
- SequenceDiagram

Condition_Mark

The Condition_Mark subject contains properties that can be used to show a state of an instance or a condition the instance chooses.

AlignConditionMarksLeft

The boolean property AlignConditionMarksLeft controls the left alignment of condition mark text. If set Checked, all newly created condition marks will have text aligned left.

- Condition marks from projects created (last saved) before Rhapsody 6.0 is aligned left.
- Alignment of existing text cannot be changed using the diagram editor

(Default = Cleared)

General

The General metaclass contains a property that controls the appearance of message parameters in sequence diagrams and in the browser.

AutoCreateExecutionOccurrence

The AutoCreateExecutionOccurrence property determines whether execution occurrences are created automatically when a message is created. See the Rational Rhapsody Help for more information on execution occurrences.

(Default = Cleared)

AutoLaunchAnimation

In general, for sequence diagrams, you have to manually select a diagram in order to have Rhapsody display the animation for the diagram (unless the diagram is already open). However, the property AutoLaunchAnimation can be set for a sequence diagram to request that Rational Rhapsody automatically launch an animated version of the diagram when the application is run in animation mode.

The property has the following possible values:

- Never - the animated diagram will not be launched automatically
- Always - the animated diagram will always be launched automatically
- If_Open - the animated diagram is launched automatically only if the sequence diagram is already open

Default = If_Open

ClassCentricMode

The ClassCentricMode property specifies whether you can create sequence diagrams with instances and messages that are not realized by model elements. When this property is set to True, you can create a class by typing Class Name>, which in turn changes the label on the instance line to :Class Name>. (Default = Cleared if its in Analysis mode, Checked if its in Design mode)

CleanupRealized

The CleanupRealized property specifies whether to delete the realized messages and classifier roles from the sequence diagram when you delete classifiers, operations, or events. (Default = Cleared if its in Analysis mode, Checked if its in Design mode)

ConfirmCreation

The ConfirmCreation property specifies whether Rational Rhapsody should confirm the creation of the corresponding operation. When you change the name of a message and this property is set to True, a dialog asks whether you want to create the operation.

When this property is set to Cleared, there is no confirmation window - the operation is created automatically. This property is relevant when the RealizeMessages property is set to Checked (usually for design mode of sequence diagrams).

(Default = Checked)

DefaultLifelineType

The property DefaultLifelineType determines whether new instance lines are of type class or file. (Default = Class)

HorizontalMessageType

The property HorizontalMessageType determines the default type of new horizontal messages: There are

4 possible values:

- Default - Same as PrimitiveOperation
- PrimitiveOperation - An operation whose body you write yourself. Rhapsody automatically generates bodies for all other types of operations.
- TriggeredOperation - A cross between an operation and an event. It is started by another object to trigger a state transition and its body is executed in response to the transition taken. Because it is a synchronous event, the sending object waits for the execution of the triggered operation.
- Event - An instantaneous occurrence that can trigger a state transition in a class.

The default is Default.

MaxNumberOfAnimMessages

The MaxNumberOfAnimMessages property specifies the maximum number of animation messages to display at any time. The property OnReachedMaxAnimMessages determines how Rhapsody behaves once this number has been reached.

(Default = 1000)

OnReachedMaxAnimMessages

The property OnReachedMaxAnimMessages determines how Rhapsody should behave when the maximum number of messages has been reached. The property can take the following values:

- Stop - Rational Rhapsody stops displaying animated messages in the diagram after the maximum number has been reached.
- KeepLast - After the maximum number of messages specified has been reached, Rational Rhapsody erases the first messages displayed. It will continue erasing displayed messages in this manner so that the number of messages displayed on the diagram at any one time does not exceed the maximum specified..

(Default = KeepLast)

RealizeMessages

The RealizeMessages property specifies whether to realize messages in sequence diagrams (use constructive mode). (Default = Checked)

SelfMessageType

The property HorizontalMessageType determines the default type of new “self” messages (messages sent from an item to itself): There are 4 possible values:

- Default - Same as PrimitiveOperation
- PrimitiveOperation - An operation whose body you write yourself. Rhapsody automatically generates bodies for all other types of operations.

- TriggeredOperation - A cross between an operation and an event. It is started by another object to trigger a state transition and its body is executed in response to the transition taken. Because it is a synchronous event, the sending object waits for the execution of the triggered operation.
- Event - An instantaneous occurrence that can trigger a state transition in a class.

(Default = Default)

ShowAnimCreateArrow

The property ShowAnimCreateArrow determines whether or not create arrows are displayed in an animated sequence diagram.

Default = Checked

ShowAnimDestroyArrow

The property ShowAnimDestroyArrow determines whether or not destroy arrows are displayed in an animated sequence diagram.

Default = Checked

ShowAnimStateMark

The property ShowAnimStateMark is used to control the display of states on animated sequence diagrams.

By default, during animation, states entered are displayed as condition marks on instance lines.

If you prefer not to have states displayed on your animated sequence diagrams, set the value of this property to False.

Default = Checked

ShowAnimTimeoutArrow

The property ShowAnimTimeoutArrow determines whether or not timeout arrows are displayed in an animated sequence diagram.

Default = Checked

ShowAnimCancelTimeoutArrow

The property ShowAnimCancelTimeoutArrow determines whether or not canceled timeout arrows are displayed in an animated sequence diagram.

Default = Checked

ShowAnimDataFlowArrow

The property ShowAnimDataFlowArrow determines whether or not data flow arrows are displayed in an animated sequence diagram to indicate the flow of data between flow ports.

Default = Checked

ShowArguments

The property ShowArguments specifies whether message arguments should be displayed in sequence diagrams, and how they should be displayed. The possible values are:

- None - Message arguments should not be displayed.
- Names - Message arguments should be displayed, but without their types.
- NamesAndTypes - Message arguments should be displayed together with their types.

Default = Names

ShowDynamicAnimInstanceName

By default, on animated sequence diagrams, Rational Rhapsody does not update the caption for an instance line if the name of the instance is changed dynamically by the application.

If you would like Rhapsody to update instance names on the diagram if they are changed, set the value of the property ShowDynamicAnimInstanceName to True.

Note that if your diagram includes the special notation that allows auto-creation of animated instances, the value of this property will have no effect. Instance names will always be updated.

Default = Cleared

ShowSequenceNumbers

The ShowSequenceNumbers property specifies whether to sequence numbers in sequence diagrams. (Default = Cleared)

SlantMessageType

The property SlantMessageType determines the default type of new “slanted” messages (messages sent from one item to another). There are 4 possible values:

- Default - Same as Event.
- PrimitiveOperation - An operation whose body you write yourself. Rhapsody automatically generates bodies for all other types of operations.
- TriggeredOperation - A cross between an operation and an event. It is started by another object to

trigger a state transition and its body is executed in response to the transition taken. Because it is a synchronous event, the sending object waits for the execution of the triggered operation.

- Event - An instantaneous occurrence that can trigger a state transition in a class.

The default is Default (Event).

InstanceLine

Contains properties that affect the display of new instance lines in sequence diagrams.

ShowStereotype

The ShowStereotype property determines whether or not classifier role stereotypes is opened when you add new instance lines to the diagram.

Default = Cleared

InteractionOperator

The InteractionOperator metaclass contains a property that controls the appearance of interaction operator guards.

ShowOperandsGuards

The ShowOperandsGuards property controls the appearance of guards. When you draw an InteractionOperator, guard may be set: it appears at the top of the IteractionOperator under the name [condition].

Setting this property to Checked displays guards. Setting this property to Cleared hides any guards.

(Default = Checked)

Message

Contains properties that affect the display of new messages in sequence diagrams.

ShowStereotype

The ShowStereotype property determines whether or not message stereotypes is opened when you add new messages to the diagram.

Default = Cleared

SequenceDiagram

The SequenceDiagram metaclass contains a property that controls the fill color of graphic elements in sequence diagrams.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

SPARK

The SPARK subject enables you to control the generation of SPARK annotations from Rational Rhapsody Developer for Ada models so they can be analyzed by the SPARK Examiner. See the Rational Rhapsody Developer for Ada documentation for detailed information. The SPARK subject contains the following metaclasses:

- Class
- Package

Class

The Class metaclass contains properties that control the examination level for the class.

ExaminerLevelBody

The ExaminerLevelBody property specifies the examination level for the class. The possible values are as follows:

- None - Do not examine the file.
- Data - Perform data-flow analysis on the file.
- Information - Perform information-flow analysis on the file.

Default = Data

ExaminerLevelSpec

The ExaminerLevelSpec property specifies the examination level for the class specification. The possible values are as follows:

- None - Do not examine the file.
- Data - Perform data-flow analysis on the file.
- Information - Perform information-flow analysis on the file.

Default = Data

Package

The Package metaclass contains properties that control the examination level for the package.

ExaminerLevelBody

The ExaminerLevelBody property specifies the examination level for the class. The possible values are as follows:

- None - Do not examine the file.
- Data - Perform data-flow analysis on the file.
- Information - Perform information-flow analysis on the file.

Default = Data

ExaminerLevelSpec

The ExaminerLevelSpec property specifies the examination level for the class specification. The possible values are as follows:

- None—Do not examine the file.
- Data - Perform data-flow analysis on the file.
- Information - Perform information-flow analysis on the file.

Default = Data

StatechartDiagram

The StatechartDiagram properties are used to control the appearance of elements in statechart diagrams.

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Bottom-Top

ButtonArray

The ButtonArray metaclass contains properties that determine the appearance and behavior of button array controls on statecharts.

ButtonFont

The ButtonFont property lets you select the font to use for the text on the face of a push button control.

To change the value of the property, click the "..." button in the box next to the property value to open the Font window. The value of the property affects both buttons already on the statechart and new buttons added to the statechart. (The display of buttons already on the statechart changes only after you refresh the diagram.)

Default = Arial 10 NoBold NoItalic

Direction

The Direction property determines whether the button array controls are used to input data, display data,

or both. The possible values are:

- In - The button arrays are only used to input data for the attribute to which it is bound.
- Out - The button arrays are only used to display data for the attribute to which it is bound.
- InOut - The button arrays are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for button array elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the button array.
- BindedElement - The name of the attribute that is bound to the button array.
- Name - The name of the button array element.
- None - No text is displayed.

Default = Name

Comment

The Comment metaclass contains a property that controls the appearance of comments in statecharts.

CommentNotation

The CommentNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of these styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Comment:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

Compartments

The Compartments property determines which of the available compartments are displayed by default for

the various types of elements.

The value for this property is a comma-delimited string containing the names of the compartments that should be visible.

Since the available compartments vary from element to element, it is recommended you not set the value of this property using the Properties window or directly in the .prp file. Use the Display Options of the element to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of these options:

- Name
- Description
- Label

Default = Description

ShowForm

Determines how note-like elements are opened. The possible values are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element

- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = Label

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

CompState

The CompState metaclass contains a property that controls the appearance of components in statecharts.

ShowCompName

The ShowCompName property specifies whether to show the component names in statecharts.

Default = Cleared

Constraint

The Constraint metaclass contains properties that specifies the constraints for statecharts.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements.

The value for this property is a comma-delimited string containing the names of the compartments that should be visible.

Since the available compartments vary from element to element, it is recommended you not set the value of this property using the Properties window or directly in the .prp file. Use the Display Options of the element to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ConstraintNotation

The ConstraintNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of these styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Constraint:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of these options:

- Name

- Description
- Label

Default = Description

ShowForm

Determines how note-like elements are opened. The possible values are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.

- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = Label

DefaultTransition

The DefaultTransition metaclass has properties that control the appearance of default transition.

line_style

The line_style property specifies the type of line used for a graphical item. The possible values are:

- straight_arrows–a straight line.
- rectilinear_arrows–rectilinear lines with right-angled corners placed at appropriate locations, depending on the start and end points of the line.
- spline_arrows–curved line without corners.

Default = spline_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = None

Depends

The Depends metaclass has properties that control the appearance of dependency relation lines in statecharts.

line_style

The line_style property specifies the type of line used for a graphical item. The possible values are:

- straight_arrows–a straight line.
- rectilinear_arrows–rectilinear lines with right-angled corners placed at appropriate locations, depending on the start and end points of the line.
- spline_arrows–curved line without corners.

Default = straight_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element

- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = None

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = Label

DigitalDisplay

The DigitalDisplay metaclass contains properties that determine the appearance and behavior of digital display controls on statecharts.

ShowName

The ShowName property determines whether or not a caption is displayed for digital display elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the digital display.
- BindedElement - The name of the attribute that is bound to the digital display.
- Name - The name of the digital display element.
- None - No text is displayed.

Default = Name

Gauge

The Gauge metaclass contains properties that determine the appearance and behavior of gauge controls on statecharts.

ShowName

The ShowName property determines whether or not a caption is displayed for gauge elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the gauge.
- BindedElement - The name of the attribute that is bound to the gauge.
- Name - The name of the gauge element.
- None - No text is displayed.

Default = Name

General

The General metaclass contains properties that specify general behavior of the statechart, such as whether to confirm deletion of objects.

DeleteConfirmation

The DeleteConfirmation property specifies whether confirmation is required before deleting a graphical element from the model. Note that this property does not apply to statechart elements, which have a separate DeleteConfirmation property. The possible values are as follows:

- Always - Rational Rhapsody displays a confirmation dialog each time you try to delete an item from the model.
- Never - Confirmation is not required to delete an element.
- WhenNeeded - Rational Rhapsody asks for confirmation if there are references to the element (or for some other reason).

Default = Never

Knob

The Knob metaclass contains properties that determine the appearance and behavior of knob controls on statecharts.

Direction

The Direction property determines whether the knob controls are used to input data, display data, or both. The possible values are:

- In - The knobs are only used to input data for the attribute to which it is bound.
- Out - The knobs are only used to display data for the attribute to which it is bound.
- InOut - The knobs are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for knob elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the knob.
- BindedElement - The name of the attribute that is bound to the knob.
- Name - The name of the knob element.
- None - No text is displayed.

Default = Name

Led

The LED metaclass contains properties that determine the appearance and behavior of LED controls on statecharts.

ShowName

The ShowName property determines whether or not a caption is displayed for LED elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the LED.
- BindedElement - The name of the attribute that is bound to the LED.
- Name - The name of the LED element.
- None - No text is displayed.

Default = Name

LevelIndicator

The LevelIndicator metaclass contains properties that determine the appearance and behavior of level

indicator controls on statecharts.

ShowName

The ShowName property determines whether or not a caption is displayed for level indicator elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the level indicator.
- Name - The name of the level indicator element.
- None - No text is displayed.

Default = Name

MatrixDisplay

The MatrixDisplay metaclass contains properties that determine the appearance and behavior of matrix display controls on statecharts.

ShowName

The ShowName property determines whether or not a caption is displayed for matrix display elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the matrix display.
- BindedElement - The name of the attribute that is bound to the matrix display.
- Name - The name of the matrix display element.
- None - No text is displayed.

Default = Name

Meter

The Meter metaclass contains properties that determine the appearance and behavior of meter controls on statecharts.

ShowName

The ShowName property determines whether or not a caption is displayed for meter elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the meter.

- BindedElement - The name of the attribute that is bound to the meter.
- Name - The name of the meter element.
- None - No text is displayed.

Default = Name

Note

The Note metaclass contains properties that specify the display of notes in statecharts.

ShowForm

Determines how note-like elements are opened. The possible values are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

OnOffSwitch

The OnOffSwitch metaclass contains properties that determine the appearance and behavior of on/off switch controls on statecharts.

Direction

The Direction property determines whether the on/off switch controls are used to input data, display data, or both. The possible values are:

- In - The on/off switches are only used to input data for the attribute to which it is bound.
- Out - The on/off switches are only used to display data for the attribute to which it is bound.
- InOut - The on/off switches are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for on/off switch elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the on/off switch.
- BindedElement - The name of the attribute that is bound to the on/off switch.
- Name - The name of the on/off switch element.
- None - No text is displayed.

Default = Name

PushButton

The PushButton metaclass contains properties that determine the appearance and behavior of push button controls on statecharts.

ButtonFont

The ButtonFont property lets you select the font to use for the text on the face of a push button control.

To change the value of the property, click the "..." button in the box next to the property value to open the Font window. The value of the property affects both buttons already on the statechart and new buttons added to the statechart. (The display of buttons already on the statechart changes only after you refresh the diagram.)

Default = Arial 10 NoBold NoItalic

ShowName

The ShowName property determines whether or not a caption is displayed for push button elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the push button.
- BindedElement - The name of the attribute that is bound to the push button.
- Name - The name of the push button element.
- None - No text is displayed.

Default = Name

Requirement

The Requirement metaclass contains properties that specify requirements in statecharts.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements.

The value for this property is a comma-delimited string containing the names of the compartments that should be visible.

Since the available compartments vary from element to element, it is recommended you not set the value of this property using the Properties window or directly in the .prp file. Use the Display Options of the element to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

RequirementNotation

The RequirementNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of these styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the three options available in the ShowForm property (Requirement:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation. This property can be set to one of these available options:

- Name
- Description
- Label

Default = Description

ShowForm

Determines how note-like elements are opened. The possible values are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = Label

SendAction

The SendAction metaclass contains properties that relate to Send Action elements in statecharts.

ShowNotation

The property ShowNotation determines what caption is opened for new Send Action elements that are added to a statechart. The property can take any of the following values:

- Name - the name of the Send Action element
- Label - the label of the Send Action element
- FullNotation - the event that is to be sent and the object that is to receive the event (target)
- Event - the event that is to be sent

This property can be set at the diagram level or higher.

Note that when you change the value of this property, the display of any new Send Action elements are affected, but the display of Send Action elements already on the diagram remains as is. (The display of existing elements on the diagram can be controlled using the Display Options... item on the context menu.)

Default = FullNotation

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = None

Slider

The Slider metaclass contains properties that determine the appearance and behavior of slider controls on statecharts.

Direction

The Direction property determines whether slider controls are used to input data, display data, or both. The possible values are:

- In - The sliders are only used to input data for the attribute to which it is bound.
- Out - The sliders are only used to display data for the attribute to which it is bound.
- InOut - The sliders are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for slider elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the slider.
- BindedElement - The name of the attribute that is bound to the slider.
- Name - The name of the slider element.
- None - No text is displayed.

Default = Name

State

The State metaclass contains properties that control the appearance of state boxes.

ShowDescription

ShowDescription is a boolean property that specifies whether or not the descriptions for the states in the statechart should be displayed.

Default = Cleared

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the

package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.

- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name_only

ShowReactions

The ShowReactions property specifies whether reactions are displayed in the corresponding states.

Default = Cleared

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = Label

StateDiagram

The StateDiagram metaclass contains properties that affect the display of statecharts.

DefaultView

The property DefaultView can be used to determine the default view for statecharts - diagram view or tabular view. This property can be set at the level of individual statecharts or higher.

Note that if the property is set at the package level or higher, it affects the display of all statecharts in the package, not just new statecharts created after the value of the property was changed.

Default = Diagram view

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

TextBox

The TextBox metaclass contains properties that determine the appearance and behavior of text box controls on statecharts.

Direction

The Direction property determines whether text box controls are used to input data, display data, or both. The possible values are:

- In - The text boxes are only used to input data for the attribute to which it is bound.
- Out - The text boxes are only used to display data for the attribute to which it is bound.
- InOut - The text boxes are used to input data and display data.

Default = InOut

ShowName

The ShowName property determines whether or not a caption is displayed for text box elements, and if so, what text should be displayed. The possible values are:

- BindedElementFullPath - The full path of the attribute that is bound to the text box.
- BindedElement - The name of the attribute that is bound to the text box.
- Name - The name of the text box element.
- None - No text is displayed.

Default = Name

Transition

The Transition metaclass contains a property that controls the appearance of transitions in statecharts.

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = spline_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram.

For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code.

The possible values are:

- Description - The content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowStereotype

The ShowStereotype property determines if, and how, a stereotype of the element is opened in a diagram. The possible values are:

- Label - The stereotype of the element is opened as a text label.
- Bitmap - The bitmap image associated with the stereotype of the element is opened.
- None - The stereotype of the element will not be displayed.

Default = None

STLContainers

Rhapsody uses containers to implement to-many relations between objects. These include relationships of one object to many, or many objects to many.

The OMContainers subject contain the following metaclasses:

- BoundedOrdered - Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.
- BoundedUnordered - Defines properties for implementing relations whose multiplicity is known and that should be accessed randomly.
- EmbeddedFixed - Defines properties for implementing embedded fixed relations.
- EmbeddedScalar - Defines properties for implementing embedded scalar (one-to-one) relations.
- Fixed - Defines properties for implementing relations of fixed size.
- General - Contains properties that enable you to set the directives and include files for the container.
- Qualified - Defines properties for implementing qualified relations, which are accessed via a key.
- Scalar - Defines properties for implementing scalar relations.
- StaticArray - Defines properties for implementing static arrays.
- UnboundedOrdered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed sequentially.
- UnboundedUnordered - Defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.
- User - Defines properties for user-defined implementations of relations.
- You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves. To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.
- For example, you can change the definition of the Implementation property as follows: Subject CG Metaclass Relation Property Implementation Enum "Default,Scalar,Fixed,BoundedOrdered, BoundedUnordered,UnboundedOrdered, UnboundedUnordered,Qualified,MyFaves, User" "Default" end end

Each property in this section includes the default value for each container type and relation type. For easier readability, the placeholder RelationType in these values represents all the other relation types that are not explicitly detailed. For example, the relation type User might have the default value of an empty string, whereas all the other relation types have the value of an empty MultiLine. The table of values would be:

BoundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->push_back($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $CType $cname.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is std::vector <$RelationTargetType>.

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is$cname->operator[]($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

The default is $cname->end().

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference.

If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

The default is <vector >,<iterator >.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() The default is $cname().

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is as follows: $IterType $iterator; $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is$IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator (Default) This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $iterator = $cname->begin().

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $iterator = $cname->begin().

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin() (Default)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various

containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator != $cname->end().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is $CType::const_iterator.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows: $CType::iterator pos = std::find($cname->begin(), $cname->end(),$item); if (pos != $cname->end()) { $cname->erase(pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear() The default is $cname->clear().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

BoundedUnordered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->push_back($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target* The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname() The default is $CType $cname.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is std::list<$RelationTargetType>.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

The default is $cname->end().

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

The default is <list>,<iterator>.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is as follows: $IterType $iterator; $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is *$iterator.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterReset.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization

cases.

The default is $IterReset.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = $cname->begin().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

(Default = $IterType)

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator != $cname->end().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is $CType::const_iterator.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a

container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows: $CType::iterator pos = std::find($cname->begin(), $cname->end(),$item); if (pos != $cname->end()) { $cname->erase(pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

The default is $cname->clear().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

EmbeddedFixed

The EmbeddedFixed metaclass defines properties for implementing embedded fixed relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $CType.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $(constant)$target $cname[$multiplicity].

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $RelationTargetType $cname[$multiplicity].

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is &$cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is ($RelationTargetType) &$cname[$index].

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator = 0;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is as follows: (($RelationTargetType)&$cname[$iterator])

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default = Blank

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $multiplicity.

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation. For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers. For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

EmbeddedScalar

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class

specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $(constant)$target.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType$reference $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is &$cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $(constRT)$target*.

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target*.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Fixed

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->insert($cname->begin(), $item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is$cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $CType $cname.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is std::vector<$RelationTargetType>.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular

type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

The default is $cname->operator[]($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

The default is $cname->end().

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

The default is < vector>,< iterator>.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is as follows: $IterType $iterator; $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is *$iterator.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterReset.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterReset.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin() The default is $iterator = $cname->begin().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator != $cname->end().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is $CType::const_iterator.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows:

$CType::iterator pos = std::find($cname->begin(), $cname->end(),$item); if (pos != $cname->end()) { $cname->erase(pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear() (Default)

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

General

The General metaclass contains properties that enable you to set the directives and include files for the container.

ContainerDirectives

The ContainerDirectives property specifies the preprocessor directives that are necessary when compiling code that uses a particular container library.

No additional directives are required when using OMContainers.

The default is as follows:

#ifdef _MSC_VER // disable Microsoft compiler warning (debug information truncated) #pragma warning(disable: 4786) #endif

ContainerIncludes

The ContainerIncludes property specifies header files that must be included when using a particular container library.

For example, when you use STLContainers, the following string causes one #include directive per container to be added to generated files (such as #include string): string, algorithm, vector, list, map, iterator Whether the #include directives are added to source or header files depends on the value of the IncludeDirective property.

The default is string,algorithm.

Qualified

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key:

$cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is as follows:

$cname->insert($CType::value_type($keyName, $item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $CType $cname.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is std::map<$keyType, $RelationTargetType>.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

The default is $cname->end().

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

The default is as follows:

($cname->find($keyName) != $cname->end() ? (*$cname->find($keyName)).second : NULL)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

The default is as follows: <map>,<iterator>,<oxf/OMValueCompare.h>

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is as follows:

$IterType $iterator; $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is (*$iterator).second.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterReset.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterReset.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin() The default is $iterator = $cname->begin().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator != $cname->end().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is $CType::const_iterator.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows:

$CType::iterator pos = std::find_if($cname->begin(), $cname->end(),OMValueCompare<const $keyType,$RelationTargetType>($item)); if (pos != $cname->end()) { $cname->erase(pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items:

$cname-clear()

The default is $cname->clear().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

The default is $cname->erase($keyName).

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

Scalar

The Scalar metaclass defines properties for implementing scalar relations.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $(constant)$target$reference.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $RelationTargetType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file.

A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $(constRT)$target$reference.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $CType.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item The default is $cname = $item.

Type

The Type property specifies the type of the container as a pointer to the relation.

StaticArray

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is as follows:

$Loop { if (!$cname[pos]) { $cname[pos] = $item; break; } }

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

The default is $CType.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to

Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is $RelationTargetType $cname[$multiplicity].

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container’s at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname[$index].

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file.
- weak - The #include directives are added to the source file with forward declarations in the header file. (Default)

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file.

A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

The default is as follows:

$Loop { $cname[pos] = NULL; }

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container:

vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is $IterType $iterator = 0;.

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is $cname[$iterator].

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterIncrement.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterIncrement.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin() The default is $iterator = 0.

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator < $multiplicity.

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is int.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the

collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

The default is as follows:

$Loop { if ($cname[pos] == $item) { $cname[pos] = NULL; } }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

SetAt

The SetAt property specifies how code is generated for the body of the mutator for a scalar container.

The default is $cname[$index] = $item.

Type

The Type property specifies the type of the container as a pointer to the relation.

UnboundedOrdered

Defines properties for implementing relations whose multiplicity is bounded and that are to be accessed sequentially.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->push_back($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target* The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname() The default is $CType $cname

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is std::vector<$RelationTargetType>.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index) The default is $cname->operator[]($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

The default is $cname->end().

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

The default is <vector>,<iterator>.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to

Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin() The default is as follows: $IterType $iterator; $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator (Default) This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterReset.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterReset.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin() The default is $iterator = $cname->begin().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator != $cname->end().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is $CType::const_iterator

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos) The default is as follows:

$CType::iterator pos = std::find($cname->begin(), $cname->end(),$item); if (pos != $cname->end()) { $cname->erase(pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear() The default is $cname->clear().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

UnboundedUnordered

The UnboundedUnordered metaclass defines properties for implementing relations whose multiplicity is unbounded (*) and that should be accessed randomly.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

The default is $cname->push_back($item).

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

The default is $cname = $CreateStatic.

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname() The default is $CType $cname.

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

The default is new $CType.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

The default is std::list>$RelationTargetType>.

Find

The Find property specifies the command used to locate an item in a container. For example, the following command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

The default is $CType $cname.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

The default is $cname.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position:

$cname-at($index)

The default is $cname->operator[]($index).

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

The default is $cname->end().

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that

describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

The default is <list>,<iterator>.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname() (Default)

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

The default is as follows:

$IterType $iterator; $IterReset

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

The default is $IterCreate.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

The default is *$iterator.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

The default is $IterReset.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

The default is $IterReset.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator's begin() operation to point the iterator to the first item in the collection:

$iterator=$cname-begin()

The default is $iterator = $cname->begin().

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

The default is $IterType.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

The default is $iterator != $cname->end().

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

The default is $CType::const_iterator.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

The default is $(constant)$target$reference.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos) The default is as follows:

$CType::iterator pos = std::find($cname->begin(), $cname->end(),$item); if (pos != $cname->end()) { $cname->erase(pos); }

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container’s clear() operation to remove all items: $cname-clear() The default is $cname->clear().

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container’s erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

User

The User metaclass defines properties for user-defined implementations of relations.

You can create your own implementations for relations by defining a new set of properties under the User metaclass. Once these are defined, you can give them permanent status by manually saving them in the factory.prp file under any other name, for example MyFaves.

To complete their installation, you must add the new name as an enumerated value to the CG::Relation::Implementation property.

Add

The Add property specifies the command used to add an item to a container.

For example, the following command calls the push_back() member function of the container class specified by $cname, and passes the item to be added as a formal parameter: $cname-push_back($item)

For maps (qualified relations), the following command inserts an item into the map based on a key: $cname-insert(map$keyType,$target*::value_type($keyName,$item))

Create

The Create property specifies the command used to create a new container.

For example, the following command allocates space for a vector of role names represented by $target and returns a pointer to the vector: new vector$target*

CreateByValue

The CreateByValue property specifies the command used to create a new container by value.

For example, the following command instantiates a list of role names represented by $target and assigns it the name stored in $cname: vector$target* $cname()

CreateStatic

The CreateStatic property defines the initialization of a relation when CG::Relation::Containment is set to Reference.

CType

The CType property specifies the collection type used to generate behaviors for relations. For example, when instantiating a vector that contains references to two clients, the vector$target* collection type determines the type of the variable cl on the left side of the assignment:

vectorClient*>* cl = new vectorClient*>(2);

In this case, the class Client is the $target of vector operations.

Find

The Find property specifies the command used to locate an item in a container. For example, the following

command calls the Find() operation for container $cname to locate the $item: $cname-find($item)

FullTypeDefinition

The FullTypeDefinition property specifies the implementation template for a typedef composite type.

See the Rational Rhapsody Help for detailed information about Composite Types.

Get

The Get property specifies a template for the code generated for the body of the accessor for a particular type of container.

For example, for a scalar relation generated in C, the body of the accessor is specified as: mecname The variable $me is replaced with the object context variable as specified by the Me property. The variable $cname is replaced with the name of the container, which is the role name for the relation.

GetAt

The GetAt property specifies a template for the name of the operation generated to retrieve one of the targets of a to-many relation using an index.

The ContainerTypes::RelationType::GetAt property specifies a template for the body of the operation. For example, the following command generates code that calls the container's at() operation to retrieve the item at the indexed position: $cname-at($index)

GetEnd

The GetEnd property specifies the command used to retrieve the last item in the container. For example, the following command calls the container's end() operation to locate the last item in the collection: $cname-end() This property and GetEndGenerate were created to adhere to the standard library convention for "finding" where iteration should end. GetEnd is generated where Get is generated. The method name is defined using the properties GetEnd and GetEndGenerate under CG::Relation.

GetKey

The GetKey property specifies a template for the name of an operation generated to retrieve an item from a map (qualified relation) based on a key.

For example, the following command retrieves an item based on the key name using the subscript operator[], which has been overloaded according to the STL definition for maps: $cname-operator[]($keyName)

IncludeDirective

The IncludeDirective property specifies how container header files are included in generated files. The property is read through a pointer to the concrete relation.

The possible values are as follows:

- strong - The #include directives are added to the header file. (Default)
- weak - The #include directives are added to the source file with forward declarations in the header file.

IncludeFiles

The IncludeFiles property enables selective framework includes of templates based on a particular relation implementation.

If this property is defined, the specified include files are added to the specification files for classes participating in a relation.

Include files can also be added to class implementation files if the container is added by reference. If the Containment property is set to Reference, a forward declaration of the container is added to the class specification file, and the #include is added to the class implementation file. A new set of properties that describe the forward declaration of the container is added to each container implementation metaclass, and the necessary modifications are made to the code generation.

Init

The Init property specifies the command used to initialize the container. For example, the following command calls the constructor for the container $cname to initialize it: $cname()

InitInCtorBody

The InitInCtorBody property specifies code to initialize containers for arrays inside the class constructor body. For example: pos=0; pos$multiplicity; pos++; $cname[pos]=NULL

InitStatic

The InitStatic property defines the initialization of a relation when CG::Relation::Containment is set to Value.

IterCreate

The IterCreate property specifies the commands used to create an iterator for traversing the items in the container. For example, the following command instantiates the iterator class and points it to the first item in the container: vector$target*::const_iterator $iterator; $iterator=$cname-begin()

IterCreateByValue

The IterCreateByValue property is the same as IterCreate, but instantiates the iterator by value.

IterGetCurrent

The IterGetCurrent property specifies the command used to retrieve the current item in a container. For example, the following command returns a pointer to the current item in the collection: *$iterator This value is the same as that for IterTest only when using the Rational Rhapsody framework container set OMContainers. When using the STL container set, an operation substituted for IterGetCurrent returns a pointer to the current item in the collection.

IterIncrement

The IterIncrement property specifies the code that increments the iterator. For example, the following command moves the $iterator ahead one item: $iterator++ (Default)

IterIncrementForCleanUp

The IterIncrementForCleanUp property specifies code to increment the iterator when a relation is cleaned inside the cleanUpRelations() method.

IterIncrementForInit

The IterIncrementForInit property specifies the code that increments the iterator, for certain initialization cases.

IterInit

The IterInit property is used to generate code that initializes an iterator over a relation. The iterator is the return value of the relation getRelation>() generated operation.

Default =

IterReset

The IterReset property specifies the command used to reset the iterator to the beginning. For example, the following command calls the iterator’s begin() operation to point the iterator to the first item in the collection: $iterator=$cname-begin()

IterReturnType

The property IterReturnType specifies the return type for functions such as getters for the various containers that Rational Rhapsody uses.

IterTest

The IterTest property specifies the command used to test whether the iterator is at the end of the set. For example, the following command returns a pointer to the last item in the collection: $iterator != $cname-end() With STLContainers, unlike OMContainers, it is possible to store a NULL value as the container. With OMContainers, IterTest retrieves the current item in the collection (the same as IterGetCurrent).

IterType

The IterType property specifies the iterator type. For example, the following command denotes that the iterator in use is the parameterized type vector>, as defined in the STL. vector$target*::const_iterator You can change the iterator type to one of your own choice.

Loop

The Loop property reuses code that performs a loop. Beginning with Version 5.0, each property in a container metaclass (for example, the OMContainers.StaticArray metaclass) can use any other property in the same metaclass. Therefore, other properties using $Loop will expand the specified Loop property body.

The default value for C++ is as follows: for (int pos = 0; pos $multiplicity; ++pos) The default value for C is as follows: int pos; for (pos = 0; pos $multiplicity; ++pos) The default value for Java is as follows: for (int pos = 0; pos $multiplicity; pos++)

Member

The Member property specifies the name of the embedded member in an embedded scalar (one-to-one) relation.

The default value for RiCContainers::EmbeddedScalar is mecname. The default value for all other subjects is $cname.

RelationTargetType

The RelationTargetType property specifies the return type for relation getters, as part of the relation implementation properties.

Remove

The Remove property specifies the command used to remove an item from a relation.

For example, the following commands call the find() operation to point the iterator to the item to item to be removed and then call erase() to remove it. vector$target*::iterator pos=find($cname-begin(), $cname-end(),$item);$cname-erase(pos) This operation applies only to “to-many” (non-scalar) containers.

For maps (qualified relations), the following commands create a pair p, whose second element, p.second, is the item to be removed. The find() operation points the iterator to the position of the pair in the collection. Finally, the item at that position is erased. pair$keyType,$target* p; p.second=$item; map$keyType,$target*::iterator pos=find($cname-begin(), $cname-end(),p); $cname-erase(pos)

RemoveAll

The RemoveAll property specifies the command used to remove all items from the container. For example, the following command calls the container's clear() operation to remove all items: $cname-clear()

RemoveKey

The RemoveKey property specifies the command generated to remove an item from a map (qualified relation) based on a key. For example, the following command calls the container's erase() operation, passing it the $keyName, which maps into a dictionary used to locate the item to be removed: $cname-erase($keyName)

Set

The Set property specifies how code is generated for the body of the mutator for a scalar container.

For example, the following command says that the container name, $cname, is a role name of the relation itself, because there is only one class involved: $cname = $item

Type

The Type property specifies the type of the container as a pointer to the relation.

TestConductor

The TestConductor subject contains properties that affect the TestConductor tool. This subject is available only if you have installed TestConductor. The TestConductor subject contains the following metaclasses:

- SequenceDiagram
- Settings

SDInstance

The SDInstance metaclass contains a number of properties that are used by TestConductor for internal data.

ExecutionIterations

This property is used by TestConductor for internal data. It should not be changed by the user.

ExecutionMode

This property is used by TestConductor for internal data. It should not be changed by the user.

ExecutionOrder

This property is used by TestConductor for internal data. It should not be changed by the user.

ParameterValues

This property is used by TestConductor for internal data. It should not be changed by the user.

SequenceDiagram

The SequenceDiagram metaclass controls sequence diagram properties used by TestConductor.

ActivationCondition

The ActivationCondition property specifies an activation condition. Activation conditions are used to specify the point in time during model execution when SD instances become activated. You can use activation conditions to model stubs or a predecessor order between several SD instances in a test definition. You can associate one activation condition with every SD. Activation conditions can specify a

starting point of SD instance simulation, such as event sending or event receiving, which in turn can be a result of the behavior defined by another SD. TestConductor supports conditional expressions for events and conditions in the following form: ObjectName->CondName(Parameters) In this syntax:

- ObjectName is a parameterized or concrete name of a class instance or an environment variable that can be represented by the system border.
- CondName is a particular kind of event, state, or method action.
- Parameters is a state of a statechart, or the name of an event or method, and the receiver of this event or method, depending on the CondName.

Rhapsody does not perform any static syntax checks on these conditions.

Default = TRUE

Parameter

The Parameter property specifies the parameterized name used in a test definition. TestConductor supports test definitions based on SDs, whose instances either have concrete or parameterized names. A parameterized name is one that is not a valid (or concrete) object name as usually used in Rational Rhapsody. You can also use anonymous class names, which do not have concrete names or parameters. In this case, the class name is internally expanded internally to the unique concrete object instance. During test execution, SDs are animated in relation to the default names. See the TestConductor Help for more information.

Default = Empty string

Settings

AcknowledgeApplyChanges

If this property is checked, TestConductor asks the user to acknowledge changes that have been made in the Edit SDInstances dialog. If cleared, changes that have been made in the dialog is accepted by TestConductor without the user being asked to acknowledge the changes.

Default = Checked

CreateTestArchitectureMode

The property CreateTestArchitectureMode controls the behavior of the TestConductor function "Create TestArchitecture".

If the value of the property is set to Standard, then each time "Create TestArchitecture is performed, TestConductor creates a component and a configuration for the newly created TestArchitecture using the default settings for components and configurations.

If the value of the property is set to Advanced, then each time "Create TestArchitecture" is performed, TestConductor displays a dialog that allows you to select which of the existing components/configurations should be used as the basis for the property values of the new component/configuration that is to be created.

Default = Standard

OverwriteTestContextDiagram

The property OverwriteTestContextDiagram determines whether existing TestContextDiagrams are overwritten when performing an #Update TestArchitecture# on a TestContext. The property can take any of the following values:

- Never - each time #Update TestArchitecture# is performed, a new TestContextDiagram is added to the existing TestContextDiagrams, i.e., existing TestContextDiagrams are never overwritten.
- askUser - each time #Update TestArchitecture# is performed, user is asked if an existing TestContextDiagram should be overwritten with the new one.
- Always - each time #Update TestArchitecture# is performed, existing TestContextDiagram is overwritten by the new one.

Default = Never

TestCase

The TestCase metaclass contains properties that affect TestConductor's behavior during TestCase execution.

AnimatedSUT

Depending on whether or not the SUT classes are animated, TestConductor uses different execution algorithms to control the execution of test cases. The property can take any of the following values:

- Automatic - TestConductor tries to deduce whether or not the SUT contains animation code, and chooses the appropriate execution algorithm.
- True - TestConductor chooses the appropriate algorithm on the assumption that the SUT classes contain animation code.
- False - TestConductor chooses the appropriate algorithm on the assumption that the SUT classes do not contain animation code.

Default = Automatic

ATGTestCase

This property is checked if the TestCase is generated by ATG, otherwise it is cleared.

CallOperationsOnlyWhenCallstackEmpty

If this property is checked, TestConductor delays operation calls that refer to inputs of TestConductor so that these operation calls are made only when the call stack of the focus thread is empty.

If the property is cleared, all operation calls are made by TestConductor immediately even if the call stack of the focus thread is not empty.

Default = Cleared

ComputeCoverage

The property ComputeCoverage determines whether or not TestConductor automatically computes and reports the model coverage achieved when executing the test cases.

Default = False

CreateSDForFailedSDInstance

If this property is checked, then for each failed SDInstance of the TestCase, a color-coded sequence diagram showing the reason for failure is added to the model when TestCase execution has finished.

Default = Cleared

ExecuteTestWithTracer

If this property is checked, tracer outputs (trace #all all) are generated during TestCase execution.

Default = Cleared

ExecutionIdleTimeout

The value entered for this property specifies the number of seconds the application must be idle before TestConductor aborts TestCase execution. If set to 0, TestConductor will not abort the TestCase.

Default = 600

MultipleConditionCheck

The property MultipleConditionCheck allows you to configure TestConductor to check the condition reached and following conditions, without system activity, until one condition mark evaluates to False. This is done by setting the value of this property to True.

Default = False

ResetAppBeforeStartTest

If this property is checked, TestConductor restarts the application each time a TestCase is executed. If the property is cleared, then if the application is already running, TestConductor executes the TestCase in the current execution state of the running application.

The property only affects sequence diagram-based TestCases. For code/flowchart/activity TestCases, TestConductor always restarts the application.

Default = Checked

TerminateAppOnQuitTest

If this property is checked, TestConductor terminates the application after TestCase execution has finished. If cleared, the application is not terminated after TestCase execution.

The property only affects sequence diagram-based TestCases. For code/flowchart/activity TestCases, TestConductor always terminates the application after TestCase execution has finished.

Default = Checked

Tolerances

This property is used by TestConductor for internal data. It should not be changed by the user.

UseOM_RETURN

The property UseOM_RETURN is used to determine how a return value is to be checked.

The property should be set to True for operations that use the animation macro OM_RETURN.

For operations that do not use the OM_RETURN macro, the value of the property should be set to False. Note that in such cases, TestConductor can only check return values for operation calls that originate from TestComponents.

Default = False

WriteTestExecutionLogFile

If this property is checked, TestConductor creates an execution log file called "C:/tmp/rtc.log". During TestCase execution, TestConductor writes log messages to this file, which can be used for purposes such as debugging.

If the machine running TestConductor does not have a directory called "C:/tmp", no log file is created.

Default = Cleared

UseCaseExtensions

The UseCaseExtensions subject contains properties that determine extensions to use cases, as described in version 1.4 of the UML standard. There is a single metaclass: Dependency. Currently, these properties are informative only - they do not affect the implementation of the model.

Dependency

The Dependency metaclass contains properties that control the extensions to use case dependencies, as defined in version 1.4 of the UML standard.

Condition

The Condition property specifies the condition applied to the extend relationship between use cases. If the condition is met, the extension is applied.

Default = Empty string

ExtensionPoint

The ExtensionPoint property specifies the extension point that is relevant for the relationship. This should correspond to one of the extension points defined for the use case (specified in the Use Case Features window).

Default = Empty string

UseCaseGe

The UseCaseGe subject contains properties that determine the default appearance of elements in use case diagrams. The metaclasses are as follows:

- Actor
- Association
- AutoPopulate
- Comment
- Complete
- Constraint
- Depends
- Flow
- Inheritance
- Note
- Package
- Requirement
- UseCase
- UseCaseDiagram

Actor

The Actor metaclass contains properties that control the appearance of actors in use case diagrams.

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed

- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Association

The Association metaclass contains properties that control the appearance of association lines in use case diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 255,0,0)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a

diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Name

ShowSourceMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "source" end of the line.

Default = Cleared

ShowSourceQualifier

The boolean property ShowSourceQualifier is used for a number of connector elements, such as Associations. When set to Checked, the source element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_0. The property can be set at the diagram level. The Display Options window can be used to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

Default = Cleared

ShowSourceRole

The boolean property ShowSourceRole is used for a variety of connector elements, such as Links and Associations. When set to Checked, the source end of the relationship is displayed alongside the connector, for example itsClass_1. The property is set at the diagram level.

Default = Cleared

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = None

ShowTargetMultiplicity

An association, link, aggregation, and composition have two ends and each end may be assigned a multiplicity number (1, 1.x, etc.). This property controls whether the assigned multiplicity number is visible (selected check box) or not (cleared check box) on the "target" end of the line.

Default = Cleared

ShowTargetQualifier

The boolean property ShowTargetQualifier is used for a number of connector elements, such as Associations. When set to Checked, the target element attribute defined as a qualifier for the association is displayed alongside the connector, for example attribute_1. The property can be set at the diagram level. The Display Options window can be used to change the qualifier show/hide setting for an individual connector. However, it does not change the value of the property at the diagram level.

Default = Cleared

ShowTargetRole

The boolean property ShowTargetRole is used for a variety of connector elements, such as Links and Associations. When set to Checked, the target end of the relationship is displayed alongside the connector, for example itsClass_2. The property is set at the diagram level.

Default = Cleared

AutoPopulate

The AutoPopulate metaclass contains properties that can be used to control the appearance of diagrams that are drawn automatically by Rhapsody.

ArrowDirection

The ArrowDirection property is used when Rhapsody automatically generates a diagram, and it determines whether the flow of connectors in the diagram runs from top to bottom or bottom to top.

There are two situations where Rhapsody automatically generates diagrams:

- If you have selected the Populate Diagrams option for Reverse Engineering (for those diagrams where this feature is supported).
- If you double-click a diagram in the browser that was generated using the Rational Rhapsody API.

Default = Bottom-Top

Comment

The Comment metaclass contains properties that control the appearance of comments in use case diagrams.

CommentNotation

The CommentNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of these styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the options available in the ShowForm property (under Comment:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and includes an ability to add compartments to that box.

Default = Note_Style

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of these available options:

- Name
- Description
- Label

If the property is set to Note_Style, then one of the options available in the ShowForm property (under Comment:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

Default = Description

ShowForm

The ShowFrom property determines how note-like elements are opened. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Complete

The metaclass Complete contains properties that determine whether or not Rational Rhapsody automatically draws the relations that exist between an element added to a diagram and elements already on the diagram.

Complete_Relation

The property Complete_Relation is used to specify that when an element is added to a diagram, Rational Rhapsody should automatically draw the relations that exist between the element and elements already on the diagram.

Default = Cleared

Constraint

The Constraint metaclass contains properties that control the constraints in use case diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

ConstraintNotation

The ConstraintNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of these styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of these options available in the ShowForm property (Constraint:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of these styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of these available options:

- Name
- Description
- Label

Default = Description

ShowForm

The ShowFrom property determines how note-like elements are opened. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The

different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Depends

The Depends metaclass contains properties that control the appearance of dependency relation lines in use case diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,0,255)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = straight_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = None

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Flow

The Flow metaclass contains properties that control how information flows are displayed in use case diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 0,147,0)

flowKeyword

The flowKeyword property is a Boolean value that specifies whether the flow keyword for the information flow is displayed in the diagram.

Default = Checked

infoItemsColor

The infoItemsColor property specifies the color used to draw information items in diagrams. (Default = 0,0,255)

line_style

The line_style property specifies the default line style for a graphical item. The possible values are as follows:

- straight_arrows - Draw a straight line.
- rectilinear_arrows - Draw a rectilinear lines with right-angled corners placed at appropriate locations, depending on the starting and ending points of the line.
- spline_arrows - Draw a curved line without corners.

Default = rectilinear_arrows

ShowConveyed

The property ShowConveyed determines whether or not flow items should be displayed alongside the flows that convey them, and if so, what text should be displayed for the flow items. The property can take any of the following values:

- Name - the name of the flow item
- Label - the label of the flow item
- None - nothing should be displayed for the flow item

Note that this property only affects the display of new flows added to a diagram. The display of flow items for flows already on a diagram can be controlled by selecting the Display Options... item from the context menu for flows.

Default = Name

Inheritance

The Inheritance metaclass contains properties that control the appearance of inheritance lines in use case diagrams.

line_style

The line_style property specifies the type of line used for a graphical item. The possible values are:

- straight_arrows - a straight line.
- rectilinear_arrows - rectilinear lines with right-angled corners placed at appropriate locations, depending on the start and end points of the line.
- spline_arrows - curved line without corners.

Default = straight_arrows

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = None

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Note

The Note metaclass contains properties that control the appearance of notes in use case diagrams.

ShowForm

The ShowFrom property determines how note-like elements are opened. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

Package

The Package metaclass contains properties that specify the appearance of packages in use case diagrams.

ShowName

The ShowName property specifies how the name of an object should be displayed. The possible values are as follows:

- Full_path - Show the object name using the full path. For example, "Default::A.B."
- Relative - Show the object name using a relative path. For example, "A.B."
- Name_only - Show only the object name without any path information. For example, "B."

Default = Name_only

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

Requirement

The Requirement metaclass contains properties that control the appearance of requirements in use case diagrams.

Compartments

The Compartments property determines which of the available compartments are displayed by default for the various types of elements. The value for this property is a comma-delimited string containing the names of the compartments that should be visible. Since the available compartments vary from element to element, it is recommended that you do not try to set the value of this property using the Properties window or directly in the .prp file. Rather, you should use an element Display Options to set which compartments are visible, and then use the Make Default option to apply these settings at the diagram or project level for new elements of this type.

Default = Empty MultiLine

RequirementNotation

The RequirementNotation property determines how annotations (Constraints/Comments/Requirements and simple notes) appear. This property can be set to one of these styles:

- Note_Style
- Box_Style

If the property is set to Note_Style, then one of the these options available in the ShowForm property (Requirement:ShowForm) can be selected: Note, Plain, or PushPin. These styles control the appearance of the annotation. The ShowForm property describes each of the three styles.

If this property is set to Box_Style, then the annotation looks like a class-box with a name compartment and an ability to add compartments to that box.

Default = Note_Style

ShowAnnotationContents

The ShowAnnotationContents property determines which text is displayed for a Note_Style annotation (Constraints/Comments/Requirements and simple notes). This property can be set to one of these available options:

- Name
- Description
- Label

Default = Description

ShowForm

The ShowFrom property determines how note-like elements are opened. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Plain - No color background behind text
- Note - Color background behind text
- Pushpin - Color background plus pin icon

Default = Note

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = Relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.
- None - Do not show stereotypes in diagrams.

Default = Label

SystemBox

The SystemBox metaclass contains properties that control the appearance of system boxes in use case diagrams.

color

The color property specifies the default color of the border of a graphical item, such as an object box. (Default = 128,0,255)

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 0,255,255

line_width

The line_width property specifies the default line width, in pixels, for drawing lines (for example, action state lines). (Default = 1)

name_color

The name_color property specifies the default color of names of graphical items. (Default = 128,128,0)

UseCase

The UseCase metaclass contains properties that control the appearance of use cases in use case diagrams.

LabelsStyle

Ordinarily, if you draw a use case element on a use case diagram, Rational Rhapsody displays the name of the use case inside the element. If you have chosen to display the label instead, then the label is opened inside the element.

This means that in cases where the label is very long, you must enlarge the element in order to have the entire label displayed.

The property LabelsStyle can be used to get around this constraint. If you change the value of the property to Caption, then the label is also displayed below the element and the text display area is automatically enlarged so that the entire label is always displayed.

Note that the value of this property also affects the display of the use case name if you have chosen to show the name rather than the label.

Default = "Default"

ShowName

The property ShowName determines the text that should be displayed next to a graphic element in a diagram. For most elements, Rational Rhapsody allows you to provide a name and a label. This allows you to provide a descriptive label in cases where the name itself may not be sufficient due to various constraints. For example, the inability to use spaces if the name of the element is to appear in the code. The possible values for this property varies for the different elements, as does the default value used. The different values used are:

- Description - the content of the description field; relevant for elements such as comments
- Full_path - the full path describing the hierarchical position of an element, for example, package_1::package_1b::class_0
- Label - the label provided for the element
- Name - the name of the element
- Name_only - the name of the element only (as opposed to the full or relative path)
- None - nothing should be displayed
- Relative - path describing the hierarchical position of an element, but only including the information that is not apparent from the depiction of the element in the diagram. For example, the name of the package containing a class is opened with the class name only if the class is not positioned inside the package in the diagram.
- Specification - the content of the specification field; relevant for elements such as constraints

Default = relative

ShowStereotype

The ShowStereotype property specifies how stereotypes are shown in UML diagrams. The possible values are as follows:

- Label - Show only the stereotype label (text).
- Bitmap - Show only the stereotype bitmap.

- None - Do not show stereotypes in diagrams.

Default = Label

UseCaseDiagram

The UseCaseDiagram metaclass contains a property that specifies the background color of a use case diagram.

Fillcolor

The Fillcolor property specifies the default fill color for the object.

Default = 218,218,218

WebComponents

The WebComponents subject controls whether Rational Rhapsody components can be managed from the Web, and specifies the necessary framework for code generation. The metaclasses are as follows:

- Attribute
- Class
- Configuration
- Event
- File
- Operation
- WebFramework

Attribute

The Attribute metaclass contains properties that determine whether attributes can be managed from the Web.

ApplyUserHelpers

If you provide a getter and/or setter for an attribute, instead of having Rational Rhapsody generate them, then the webify code generated uses these getters/setters. If for some reason you do not want Rational Rhapsody to use the user-provided getters/setters in the webify code, you can set the value of the property ApplyUserHelpers to False.

If the property is set to False, Rational Rhapsody will not use the user-provided getters/setters, nor will it autogenerate getters/setters for this purpose.

Default = Checked

WebManaged

The WebManaged property is a Boolean value that specifies whether the element can be managed from the Web. This property can be used as an alternative to stereotyping an element.

Default = Cleared

Class

The Class metaclass contains properties that determine whether classes can be managed from the Web.

WebifyFullClassName

The string used in the code to register webified elements cannot exceed 64 characters. To overcome this limitation, you can set the value of the property WebifyFullClassName to False, and then Rational Rhapsody uses only the class name rather than the full name of the class (including namespaces).

Default = Checked

WebifyPropagateLinks

If you specify a class to be WebManaged, classes associated with the class will also contain calls to webify operations. If, however, such associated classes are not specified as WebManaged, then these calls will result in compilation errors. In such cases, you can prevent Rational Rhapsody from generating calls to webify operations in the associated classes by setting the value of the property WebifyPropagateLinks to False.

Default = Checked

WebManaged

The WebManaged property is a Boolean value that specifies whether the element can be managed from the Web. This property can be used as an alternative to stereotyping an element.

Default = Cleared

Configuration

The Configuration metaclass contains properties that specify the configuration settings for the model.

CommunicationLayerScheme

The CommunicationLayerScheme property specifies which communication layer to use with the Webify Toolkit. The possible values are as follows:

- Auto - Leave the program's default value. Currently, this is SUN Java.
- JavaScript - Make sure the communication works with JavaScript.
- Microsoft VM - Java communication layer. There is a cosmetic advantage to this option.

Change the property value to JavaScript if you have the problem with an empty right pane, as described in the Known Limitations section of the Release Notes.

Default = Auto

Note: Because Microsoft no longer supports VM in the IE environment, the Webify feature uses SUN

Java by default.

This affects any models with the property WebComponents::Confiuration::CommunicationlayerScheme that is set to Auto or Microsoft VM.

- If this property is set to JavaScript, then there should be no change.
- If the above property is set to Auto or MicrosoftVM, then you need to be sure that your Sun's Java is enabled in Internet Explorer.

Note that the required VM version is 1.4.2.04 or higher.

HomePageURL

The HomePageURL property specifies the URL to the home page. This setting corresponds to the home page attribute defined in the Advanced Webify Toolkit Settings window.

Default = cgibin?Abs_App=Abstract_Default

Port

The Port property specifies the server port. This setting corresponds to the server port attribute defined in the Advanced Webify Toolkit Settings window.

Default = 80

RefreshPeriod

The RefreshPeriod property specifies the refresh timeout. This setting corresponds to the refresh period parameter defined in the Advanced Webify Toolkit Settings window.

Default = 1000

SignaturePageURL

The SignaturePageURL property specifies the URL to the signature page. This setting corresponds to the signature page attribute defined in the Advanced Webify Toolkit Settings window.

Default = sign.htm

Event

The Event metaclass contains a property that determines whether events can be managed from the Web.

WebManaged

The WebManaged property is a Boolean value that specifies whether the element can be managed from the Web. This property can be used as an alternative to stereotyping an element.

Default = Cleared

File

The File metaclass contains a property that determines whether files can be managed from the Web.

WebManaged

The WebManaged property is a Boolean value that specifies whether the element can be managed from the Web. This property can be used as an alternative to stereotyping an element.

Default = Cleared

Operation

The Operation metaclass contains a property that determines whether operations can be managed from the Web.

WebManaged

The WebManaged property is a Boolean value that specifies whether the element can be managed from the Web. This property can be used as an alternative to stereotyping an element.

Default = Cleared

WebFramework

The WebFramework metaclass contains properties that control the instrumentation code generated for Web-enabled elements.

GenerateInstrumentationCode

The GenerateInstrumentationCode property is a Boolean value that determines whether code generation for the corresponding configuration is enabled. The value of this property corresponds to the Web Instrumentation checkbox on the Settings page for a configuration.

Default = Cleared

WebInstrumentationIncludes

The property WebInstrumentationIncludes is used to specify the dependencies that must be included for elements that are web-enabled.

Default = WebComponents/WebComponentsTypes.h (C, C++), com.ibm.rational.rhapsody.webComponents. (Java)*

WSDL

The WSDL subject contains properties that support Web Service Description Language. It contains a single metaclass: Package.

Package

The Package metaclass contains properties that support WSDL (Web Service Description Language).

Namespaces

The Namespaces property is used when generating WSDL specification file from "services" stereotyped model (in Rational Rhapsody using the NetCentric profile). In addition, the Namespaces property defines the XML namespace used within the WSDL file. Namespaces identify where the data types used in the XML file are defined. You can enter a string or a URL.

Default = xsd=http://www.w3.org/2001/XMLSchema soap=http://schemas.xmlsoap.org/wsdl/soap/

TargetNamespace

The TargetNamespace property is used when generating WSDL specification file from "services" stereotyped model (in Rational Rhapsody using the NetCentric profile). it allows the user to define the "targetNamespace" attribute of the "definition" WSDL tag. The "targetNamespace" is an XML attribute. Here is where newly created elements and attributes reside.

Default = http://www.yourCompanyName.com/yourProductName/

XSD

The XSD subject contains properties that support XML Schema Definition. It contains a single metaclass: Type.

Type

The Type metaclass contains properties that support XSD (XML Schema Definition).

ImpXSDType

The ImpXSDType property is used when generating WSDL specification file from "services" stereotyped model. When a user designs a "services" stereotyped model in Rational Rhapsody, they will not be using any "XSD" types. They is using the usual types such as int, char, and so forth. When they generate a WSDL specification from their model, these types should be mapped to XSD types that are already being modeled in Rational Rhapsody in the NetCentric profile. The mapping is being done through the ImpXSDType property.

Default = Empty string