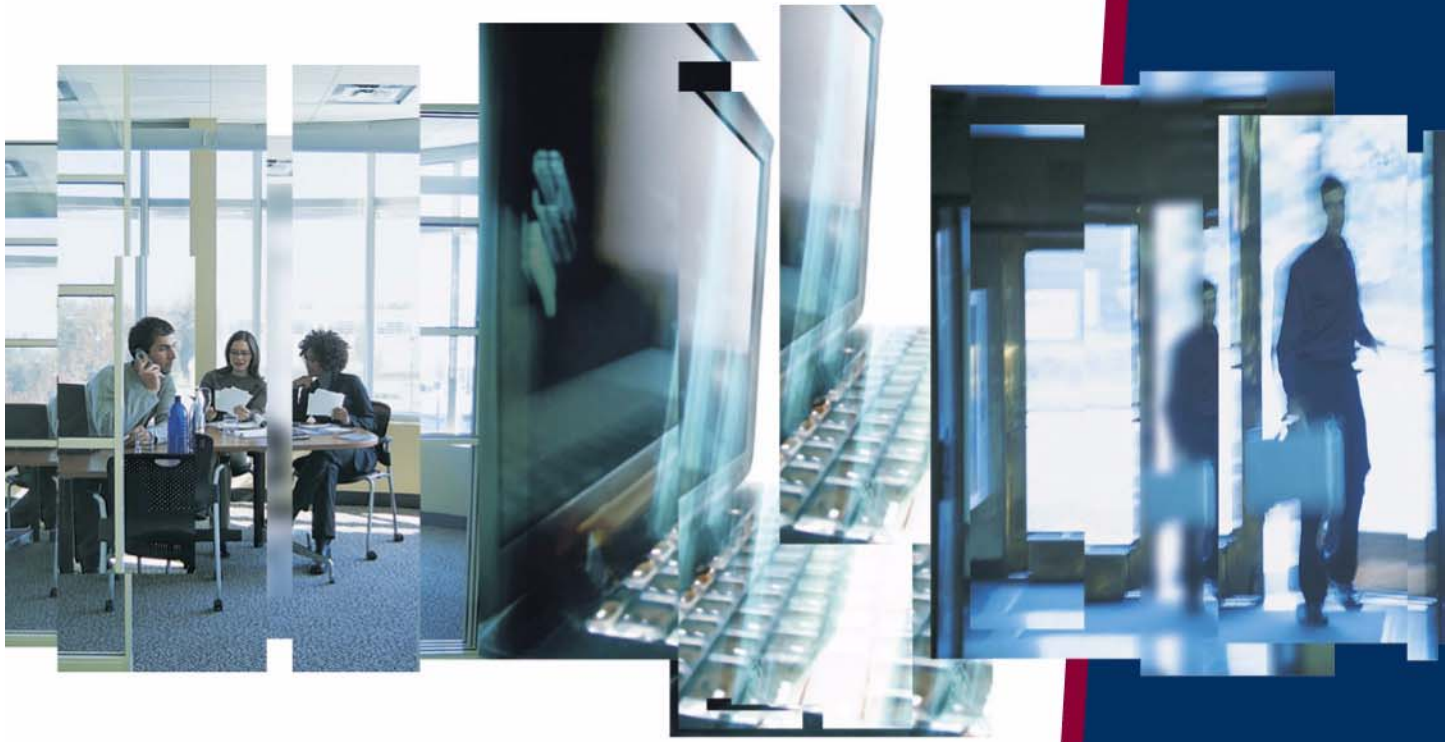
Telelogic

Rhapsody

Properties Reference Manual



IBM®

Rhapsody®

Properties Reference Manual

This edition applies to Telelogic Rhapsody 7.4 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Basic Concepts

As an open tool, Rhapsody provides a high degree of flexibility in how you set up the visual programming environment (VPE). Project properties are name-value pairs that enable you to customize many aspects of environment interaction and code generation. You can set existing properties, or create whole sets of new ones, to tailor Rhapsody to your particular needs.

This section contains information on the following topics:

Note

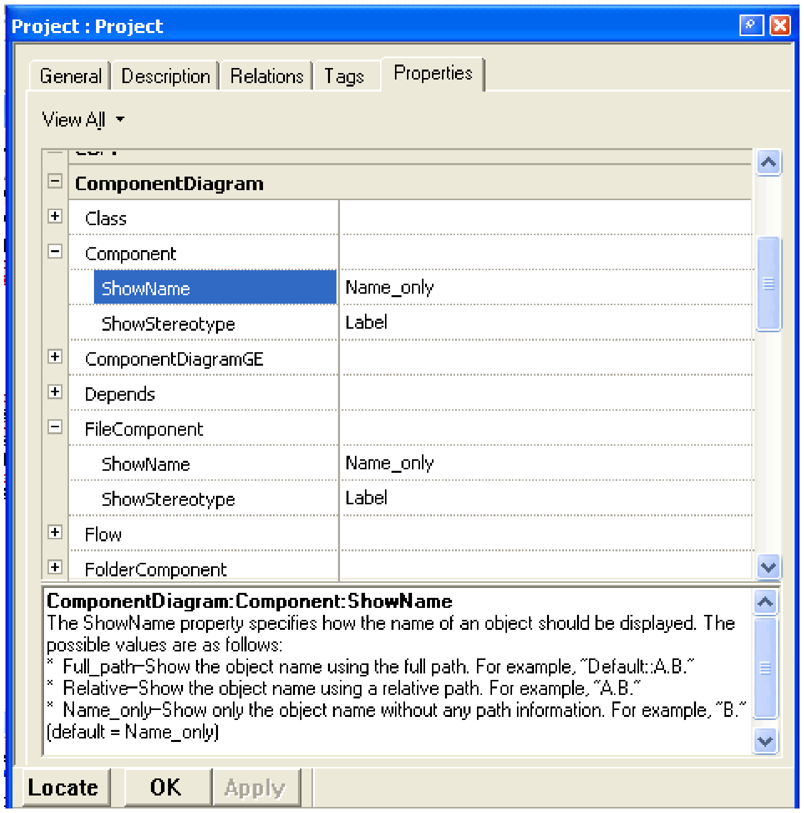
This book contains generic information about Rhapsody’s properties and how to work with them. If you want to find out what a specific property does, see *Rhapsody Property Definitions*, which is accessible from the *List of Books*.

Rhapsody Properties

Properties are user-defined, tagged values that can be attached to any modeling element. You can think of each element as having its own set of properties. Rhapsody tools, such as the code generator, reference many properties. You can modify properties to customize the tool to work in a certain way, such as setting the default color of a state box in a statechart. You can change properties at the site, diagram, package, configuration, or class level—or even at the individual operation or attribute level. Only properties that are relevant for a particular element are accessible from that element. The element on which you set a property determines its effectiveness. In other words, setting a property for a configuration provides a default for elements in the configuration. Precedence goes to the element with the lowest level of granularity. In other words, properties explicitly defined for an individual operation would override those set at the project level.

Property Groups and Definitions

The Rhapsody properties are classified according to *subject* and *metaclass* with the individual property names listed under each metaclass. Selecting a subject, metaclass, or property name listed in the Features dialog box displays the definition of the selected item below, as shown here.



ComponentDiagram	
Class	
Component	
ShowName	Name_only
ShowStereotype	Label
ComponentDiagramGE	
Depends	
FileComponent	
ShowName	Name_only
ShowStereotype	Label
Flow	
FolderComponent	

ComponentDiagram:Component: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)

The right column indicates the type of information in the property value. The possible values are as follows: Metaclasses are listed in alphabetical order under each subject.

- `Enum`
- `Bool`
- `String`
- `MultiLine`
- `Color` (RGB value)
- `Int`
- `Double`
- `Font`
- `File`

See **Setting Rhapsody Properties** for information on changing property values.

Subjects

The following table lists the Rhapsody subjects.

Subject	Description
Activity_diagram	Controls the appearance of activity diagrams.
Animation	Controls the behavior of black box animation.
ATL	Controls ATL classes. This subject applies only to Rhapsody in C++.
Browser	Controls the information displayed in the Rhapsody browser.
CG	Controls how code is generated. These properties are language-independent.
Collaboration_Diagram	Controls the appearance of collaboration diagrams
COM	Controls how mixed, distributed applications and objects find and interact with each other over a network. This subject applies only to Rhapsody in C++.
ComponentDiagram	Controls the appearance of component diagrams.
ConfigurationManagement	Defines the command strings needed by various configuration management (CM) tools to interface with Rhapsody.
<ContainerTypes>	Controls how items stored in containers are accessed and manipulated. The subject names are `Java(1.1)Containers`, `Java(1.2)Containers`, `OMContainers`, `OMCorba2CorbaContainers`, `OMCpp2CorbaContainers`, `OMCppOfCorbaContainers`, `OMUContainers`, `RiCContainers`, or `STLContainers`, depending on your programming language and environment.
CORBA	Controls how CORBA interacts with Rhapsody. This subject applies only to Rhapsody in C++.
DeploymentDiagram	Controls the appearance of deployment diagrams.
DiagramPrintSettings	Controls how diagrams are printed.
Dialog	Controls which properties are displayed in the Properties tab
General	Controls the general aspects of the Rhapsody display.
IntelliVisor	Controls the IntelliVisor feature.
<lang>_CG	Controls the language-specific aspects of code generation. The subject name is `C_CG`, `CPP_CG`, or `JAVA_CG`.
<lang>_ReverseEngineering	Controls how Rhapsody imports legacy code. The subject name is `C_ReverseEngineering`, `CPP_ReverseEngineering`, or `JAVA_ReverseEngineering`.

Subject	Description
<lang>_Roundtrip	Controls how changes made to the model are roundtripped to the code, and vice versa. The subject name is `C_Roundtrip`, `CPP_Roundtrip`, or `JAVA_Roundtrip`.
`Model`	Controls prefixes added to attributes, variables, and arguments to reflect their type.
ObjectModelGe	Controls the appearance of object model diagrams (OMDs).
QoS	Provides performance and timing information.
ReverseEngineering	Controls how Rhapsody deals with legacy code.
RoseInterface	Controls how Rhapsody imports models from Rational Rose® 98 or 2000.
RTInterface	Controls how Rhapsody interacts with requirements traceability tools.
SequenceDiagram	Controls the appearance of sequence diagrams.
SPARK	Enables you to control the generation of SPARK annotations from Rhapsody in Ada models so they can be analyzed by the SPARK Examiner.
Statechart	Controls the appearance of statecharts.
TestConductor	Controls contains properties that affect the TestConductor™ tool.
UseCaseExtensions	Controls extended UCDs.
UseCaseGe	Controls the appearance of use case diagrams (UCDs).
WebComponents	Controls whether Rhapsody components can be managed from the Web, and specifies the necessary framework for code generation.

Metaclasses

Under each subject, Rhapsody lists *metaclasses*. Metaclasses define properties for groups of things, such as attributes, classes, and configurations.

For example, under the `Statechart` subject and the `State` metaclass, the properties `color`, `line_width`, and `name_color` determine the default color and line width of state boxes and the text color of state names. The notation is `Subject::Metaclass::property`; for example, `Statechart::State::color`. Note that you can always change the properties of an element in a statechart or diagram on-the-fly—project properties specify the default appearance.

Regular Expressions

Many properties use regular expressions to define valid command strings. For example, the property **ParseErrorMessage** uses the following regular expression for Microsoft® environments:

```
([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)
```

This expression defines the rules used to parse error messages on Microsoft systems. If you redefine properties that require regular expressions, you must use the correct expression syntax.

Regular Expression Syntax

Regular expression syntax is defined[1] as a “regular expression is zero or more branches, separated by |. It matches anything that matches one of the branches. A branch is zero or more pieces, concatenated. It matches a match for the first, followed by a match for the second, etc.”

- A piece is an atom possibly followed by `*`, `+`, or `?`.
- An atom followed by `*` matches a sequence of 0 or more matches of the atom.

  For example, the atom `.*` matches zero or more instances of any character (a period matches any character).
- An atom followed by `+` matches a sequence of 1 or more matches of the atom.

  For example, the atom `.+` matches one or more instances of any character.
- An atom followed by `?` matches a match of the atom, or the null string.

  For example, the atom `.?` matches a single character or the null string, such as at the end of an input string.
- An atom is a regular expression in parentheses (matching a match for the regular expression), a range, or:
  - `.` (matching any single character)
  - `^` (matching the beginning of the input string)
  - `$` (matching the end of the input string)
  - A \ followed by a single character (matching that character)
  - A single character with no other significance (matching that character)

Consider the following regular expression:

```
([a-zA-Z_][a-zA-Z0-9_]*)
```

1.Copyright (c) 1986 by U. of Toronto. Written by Henry Spencer.

This regular expression, enclosed in parentheses, matches a sequence of two ranges—any single uppercase or lowercase letter, or underscore character; followed by zero or more uppercase or lowercase letters, digits 0-9, or the underscore character.

Parsing Regular Expressions

Parts of the expression contained within parentheses are called *tokens*.

The regular expression for the **ParseErrorMessage** property is as follows:

```
([^(]+)[(]([0-9]+)[)] [:] (error|warning|fatal error)
```

It consists of the following parts:

- `([^(]+)`—This is the first token. The caret at the beginning of the set is a `NOT` operator that matches any character except those in the set. This token tells the parser to ignore all characters until the first occurrence of an open parenthesis.
- `[(]`—The parser should search for exactly one opening parenthesis.
- `([0-9]+)`—This is the second token It tells the parser to search for a sequence of one or more digits in the range of 0 to 9.
- `[)]`—The parser should search for exactly one closing parenthesis.
- `[:]`—The parser should search for exactly one colon.
- `(error|warning|fatal error}`—This is the third token. It tells the parser to search for one of the strings "error," "warning," or "fatal error."

The property **ErrorMessageTokensFormat** works with **ParseErrorMessage** to determine how many tokens can be contained in an error message, and the relative positions in the message string of tokens that represent the file name and line number of the error, respectively. The second token in the sample regular expression would most likely represent a line number, depending on how **ErrorMessageTokensFormat** was defined.

Based on this regular expression, the parser would interpret the string "`(3457):warning`" as a valid error message indicating a warning condition at line 3457 in the program.

Property File Format

All the property (`*.prp`) files use an `LL1` syntax for a simple, recursive descent parser. The parser currently has no error recovery and effectively stops at the first error. Tokens enclosed within curly braces {} are optional. Those enclosed within angle brackets <> are further decomposed according to their own BNF (Backus Naur Form) descriptions.

The BNF for the `*.prp` files is as follows:

```
<file> ::= {"Subject" <subject>} "end"
```

For example, the `factory.prp` file begins with an optional list of subjects, each beginning with the keyword “`Subject`,” and ends with the required keyword “`end`”:

```
Subject General
Subject Statechart
Subject ObjectModelGe
.
.
.
end
```

```
<subject> ::= <name> {"Metaclass" <metaclass>} "end"
```

As another example, the subject `General` begins with a name, followed by a list of metaclasses, followed by the keyword “end”:

```
Subject General
Metaclass Graphics
Metaclass Model
end
```

```
<metaclass> ::= <name> {"Property" <property>} "end"
```

The file contains the following type declarations:

- “`Bool`”

A string that indicates a type with two possible values, `TRUE` or `FALSE.`

- `<enum values> ::= <quoted string>`

The enum values string is a comma-separated list of legal, enumerated values. A second quoted string indicates the default. For example, the quoted string “on,off” contains enumerated values.

- `<value> ::= <quoted string>`

A value. For example, a property value could be the quoted string “`Arial 10 NoBold NoItalic`”.

- `<quoted string> ::= <quote> <escaped chars> <quote>`

A quoted string is a string that starts and ends with double-quotes and can contain newlines. A backslash must precede any literal double-quote or backslash characters within the string. For example, “`FALSE`” is a quoted string.

Rhapsody Keywords

Many properties reference other properties, using the `$` symbol. For example, the command string for the property `ConfigurationManagement::ClearCase::AddMember` begins as follows:

```
"$OMROOT/etc/Executer.exe"
```

This substring references the predefined variable `OMROOT`, set in your `rhapsody.ini` file to the location of the `Share` directory in the Rhapsody installation. Expanded, this string becomes:

```
"<install_dir>\Share/etc/Executer.exe"
```

For a description, see **The Executer**.

Keywords are used in the following areas:

- Makefile generation
- Standard operations
- Relation implementation properties
- Names of generated operations
- Headers and footers
- Configuration management

The Table lists the predefined variables used in Rhapsody.

Keywords	Where Used	Description
`$archive`	`ConfigurationManagement`	The file name (including the full path) of the archive that you selected in the Connect to Archive dialog box. This can be either a file or a directory.
`$archiveddirectory`	`ConfigurationManagement`	The directory part of `$archive`. If `$archive` is a directory, `$archive` and `$archiveddirectory` are the same.
`$arguments`	`ATL`	The arguments of the operation.
`$Arguments`	`<lang>_CG`	The event or operation argument’s description, used by the **DescriptionTemplate** property.

Keywords	Where Used	Description
`$attribute`	`CG`	The object of an operation on attributes. The qualifier `:c` capitalizes the name of the attribute.
`$base`	`<lang>_CG`	The name of the reactive object.
`$CheckOut`	`ConfigurationManagement`	The command executed to check configuration items out of the archive using the main Configuration Items dialog box.
`$class`	`ATL`	The name of the ATL class.
`$ClassClean`	Makefiles	The list of class files used in a build.
`$cname`	`CG, <Container Types>, <lang>_CG`	The name of the container used to hold relations. Typical containers are arrays, lists, stacks, heaps, and maps.
`$coclass`	`ATL`	The name of the coclass that exposes the COM interface.
`$CodeGeneratedDate`	`CG, <lang>_CG`	The date of code generation. This information is printed in the headers and footers of generated files.
`$component`	`ATL`	The name of the component,
`$ComponentName`	`CG`	The name of the component that caused the code to be generated. This information is printed in the headers and footers of generated files.
`$ConfigurationName`	`CG, <lang>_CG`	The name of the configuration that caused the generation of the model element found in a file. This information is printed in headers and footers of generated files.
`$datamem`	`ATL`	The data member.
`$DeclarationModifier`	`ATL`	The declaration modifier.
`$Description`	`<lang>_CG`	The element description, used by the property **DescriptionTemplate**.
`$Direction`	`<lang>_CG`	The argument direction (`in`, `out`, and so on), used by the **DescriptionTemplate** property.
`$dupinterface`	`ATL`	The name of the duplicate interface.
`$executable`	`<lang>_CG`	The path to the executable binary file generated by the Rhapsody code generator.

Keywords	Where Used	Description
`$FILENAME`	`CPP_CG`	The name of the file used: • To generate source code for individual classes to user-specified directories • To specify that a statement should not be imported during reverse engineering (the #ifndef that protects h files from multiple includes)
`$Fork`	Framework: `start` method	Used to specify whether the `OMMainThread` singleton event loop should run on the application main thread or in a separate thread.
`$FullCodeGeneratedFileName`	`CG, <lang>_CG`	The full path name of the file. This information is printed in headers and footers of generated files.
`$FULLFILENAME`	`CG`	The full name of the file used: • To generate source code for individual classes to user-specified directories • To specify that a statement should not be imported during reverse engineering (the `#ifndef` that protects h files from multiple includes)
`$FullModelElementName`	`CG, <lang>_CG`	The full name of a model element in `<package>::<class>` format. is printed in headers and footers of generated files. For example, `Radar::Engine`, for a class named `Engine` found in a package named `Radar`.
`$FullName`	`<lang>_CG`	The full path of the element `(P1::P2::C.a)` used by the **DescriptionTemplate** property.
`$id`	`ATL`	The identifier.
`$IDInterface`	`ATL`	The interface ID of a COM interface.
`$index`	`<Container Types>`	An index used to randomly access items in a container.
`$instance`	Property `CORBA::TAO::InitialInstance`	Refers to the default initial instance of the TAO ORB.
`$interface`	`ATL`	The name of the interface.

Keywords	Where Used	Description
`$interfaceSeq`	Property: `CORBA::Class::IDLSequence`	Represents the name of the CORBA interface with the string *Seq* added to the end of the term.
`$item`	`CG, <Container Types>`	A class or instance whose behaviors are implemented by a container. Rhapsody generates various `add`, `remove`, `find`, and `get` operations to manipulate items in containers.
`$iterator`	`<Container Types>`	The name of the iterator used to traverse a container.
`$keyname`	`<Container Types>`	The name of a key used to access items in maps. A key is usually a string that maps to a dictionary that is used to locate items.
`$label`	`ConfigurationManagement`	An optional revision label of a configuration item, provided in the Check In/Check Out dialog box.
`$log`	`ConfigurationManagement`	An optional comment provided in the Check In dialog box.
`$LogPart`	`ConfigurationManagement`	The user-specified comment for the CM operation.
`$Login`	`CG, <lang>_CG`	The login name of the user who generated the file. This information is printed in headers and footers of generated files.
`$makefile`	`<lang>_CG`	The name of the makefile generated by the Rhapsody code generator.
`$maketarget`	`<lang>_CG`	Depending on the option selected in the Code menu, this expands to the one of the following: • Build • Clean • Rebuild
`$member`	`<lang>_CG`	The name of the reactive member (equivalent to the base class) of the object.
`$mePtr`	`<lang>_CG`	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.

Keywords	Where Used	Description
`$mode`	`ConfigurationManagement`	A flag indicating the locking mode provided in the Check In/Check Out dialog box. If the item is locked, `$mode` is replaced with the contents of the CM property `ReadWrite`. If unlocked, `$mode` is replaced with the contents of the property `ReadOnly`.
`$ModePart`	`ConfigurationManagement`	The locking mode of the CM operation. For example, you can check out a file from an archive as either locked or unlocked.
`$Name`	`<lang>_CG`	The element name, used by the **DescriptionTemplate** property.
`$noOfArgs`	`ATL`	The number of arguments for the operation.
`$OMAdditionalObjs`	Makefiles	The list of files to be included in the executable.
`$OMAllDependencyRule`	Makefiles	The dependency rule of a specific source file (`A.cpp: A.h B.h C.idl`).
`$OMBuildSet`	Makefiles	The compiler switches for Debug versus Release mode, as specified in the Settings dialog box for the active configuration.
`$OMCleanOBJS`	Makefiles	The list of delete commands for each object file in the makefile. Each entry in the list is created from the value of the **ObjCleanCommand** property.
`$OMCOM`	Makefiles	Specifies that the COM application to be linked is a windows application rather than a console application. This keyword is resolved based on the value of the `<lang>_CG::<Environment>::COM` property.
`$OMConfigurationCPPCompileSwitches`	Makefiles	The compiler switches specified by the **CompileSwitches** property for a configuration.
`$OMConfigurationLinkSwitches`	Makefiles	The link switches of the configuration, set in the Settings tab for the configuration.

Keywords	Where Used	Description
`$OMContextDependencies`	Makefiles	The list of dependencies and the compilation command for each model file that should be built as part of the component. Each entry is made up of the value of the **DependencyRule** property followed by the value of the **CPPCompileCommand** property.
`$OMContextMacros`	Makefiles	The set of generated macros, including: • `OMROOT` • `CPP_EXT/C_EXT` • `H_EXT` • `OBJ_EXT` • `LIB_EXT` • `INSTRUMENTATION` • `TIME_MODEL` • `TARGET_TYPE` • `TARGET_NAME` • `The “all” rule` • `TARGET_MAIN` • `LIBS` • `INCLUDE_PATH` • `ADDITIONAL_OBJS` • `OBJS` See **MakeFileContent** for more information.
`$OMCPPCompileCommandSet`	Makefiles	The compilation switches related to the **CPPCompileDebug**/ **CPPCompileRelease** properties. The property to be used is based on the value of the **BuildCommandSet** property. Set the value of **BuildCommandSet** using the configuration Settings tab in the browser.
`$OMCPPCompileDebug`	Makefiles	The compile switches needed to create a Debug version of a component in a given environment, as specified by the **CPPCompileDebug** property.

Keywords	Where Used	Description
`$OMCPPCompileRelease`	Makefiles	The compile switches needed to create a Release version of a component in a given environment, as specified by the **CPPCompileRelease** property.
`$OMFileCPPCompileSwitches`	Makefiles	This is used in the property **CPPCompileCommand** to bring in additional GUI-defined settings. The content is generated by Rhapsody (either based on content of fields or based on internal rules). It is one of the predefined keywords including, but not limited to: • `$OMCPPCompileDebug` • `$OMCPPCompileRelease` • `$OMLinkDebug` • `$OMLinkRelease` • `$OMBuildSet` • `$OMContextMacros`
`$OMDefaultSpecification Directory`	Makefiles	Supports the default specification/ implementation source directory feature. To set a default directory for a configuration, set the `<lang>_CG::Configuration::DefaultSpecification-Directory and <lang>_CG::Configuration::DefaultImplementation-Directory` properties
`$OMDEFExtension`	Makefiles	The extension of the definition file (`.def`). This keyword applies to the MicrosoftDLL/COM environments.
`$OMDllExtension`	Makefiles	The extension of the dynamic linked library file (`.dll`). This keyword applies to the MicrosoftDLL/COM environments.
`$OMExeExt`	Makefiles	The extension of the compiled executable.
`$OMFileDependencies`	Makefiles	Used as part of a source file dependency line. It is a calculated list of files on which the source file depends.

Keywords	Where Used	Description
`$OMFileImpPath`	Makefiles	The relative name and path of the implementation file. It is used in a source file dependency and compilation commands.
`$OMFileObjPath`	Makefiles	The relative path and name of an object file that is related to a given implementation and specification files. It is used as part of a file compilation command.
`$OMFileSpecPath`	Makefiles	The relative path and name of a specification file. It is used in a source file dependency line.
`$OMFlagsFile`	Makefiles	Maintained for backwards compatibility.
`$OMImpIncludeInElements`	Makefiles	The list of all `#includes` done in the related implementation file. It is used as part of a source file dependency line.
`$OMImplExt`	Makefiles	The extension of an implementation file generated for a model element.
`$OMIncludePath`	Makefiles	The include path. The path is calculated from dependencies between components and from the **Include Path** setting in the active component/configuration feature dialog box.
`$OMInstrumentation`	Makefiles	The active configuration instrumentation mode (*None*, *Tracing*, or *Animation*).
`$OMInstrumentationFlags`	Makefiles	Represents the preprocessor directives required for the selected type of instrumentation: animation, tracing, or none.
`$OMInstrumentationLibs`	Makefiles	Represents the libraries required for the selected type of instrumentation: animation, tracing, or none.
`$OMLibExt`	Makefiles	The extension of library files.
`$OMLibs`	Makefiles	The names of additional libraries (besides the framework library) to link when building a component. It is calculated from dependencies between components and the **Libraries** list in the active component/ configuration feature dialog boxes.

Keywords	Where Used	Description
`$OMLibSuffix`	Code Generation	Represents the suffix to use for library names. The keyword is replaced by the value of the property DebugLibSuffix or the property ReleaseLibSuffix depending upon the build.
`$OMLinkCommandSet`	Makefiles	The link switches related to the **LinkDebug**/**LinkRelease** properties. The property to be used is based on the value of the **BuildCommandSet** property. Set the value of **BuildCommandSet** using the configuration Settings tab in the browser.
`$OMLinkDebug`	Makefiles	The environment-specific link switches used to build a Debug version of a component. This is the value of the `LinkDebug` property.
`$OMLinkRelease`	Makefiles	The value of the **LinkRelease** property.
`$OMMainImplementationFile`	Makefiles	The main file name and path: [<imp dir>/]$TARGET_ MAIN)$(CPP_EXT)
`$OMMakefileName`	Makefiles	The name of the make file.
`$OMModelLibs`	Makefiles	The library component the model depends on. For example, if executable component A depends on the library component `L`, this keyword is replaced with the string `<filepath>\L.lib`.
`$OMObjExt`	Makefiles	The extension of object files (temporary compiler files) for a given environment. This is the value of the **ObjExtension** property.
`$OMObjs`	Makefiles	The list of object files to link into the build by the makefile.
`$OMObjectsDir`	Makefiles	A calculated keyword based on the property `<lang>_CG::<Environment>::ObjectsDirectory`).
`$OMROOT`	`ConfigurationManagement, General, <lang>_CG, <lang>_Roundtrip,` makefiles	The location of the `\Share` subdirectory in the Rhapsody installation. This is set in your `rhapsody.ini` file.

Keywords	Where Used	Description
`$OMRPFrameWorkDll`	Makefiles	Links the COM application with the DLL version of the framework instead of the default static libraries. This keyword is resolved based on the value of the `<lang>_CG::<Environment>::`**RPFrameWorkDll** property.
`$OMRulesFile`		Maintained for backwards compatibility.
`$OMSourceFileList`	Makefiles	(Rhapsody in J) Lists the source (`*.java`) files used in a build.
`$OMSpecExt`	Makefiles	The extension of the specification file generated for a model element.
`$OMSpecIncludeInElements`	Makefiles	Lists all the `#includes` done in the related specification file.
`$OMSubSystem`	Makefiles	The type of program for the Microsoft linker (for example, windows).
`$OMTargetMain`	Makefiles	The name of the file that contains the `main()` function for an executable component.
`$OMTargetName`	Makefiles	The name of the compiled version of a component.
`$OMTargetType`	Makefiles	The type of component to be built (library or executable),
`$OMTimeModel`	Makefiles	The time model setting for a configuration (simulated or real time).
`$OMUserIncludePath`	INTEGRITY build files (.gpj)	Represents the content of the Include Path field found on the Settings tab of the Features dialog box for configurations. This content is included in generated .gpj files for environments that use such files, for example, INTEGRITY5.
`$operations`	`ATL`	The list of operations.
`$opname`	`ATL`	The name of the operation.
`$opRetType`	`ATL`	The return type of the operation.
`$package`	`ATL`	The name of the package.
`$PackageLib`	`ATL`	The package library.
`$ProgID`	`ATL`	The value of the `ProgID` property (Default = `$component.$class.1`).
`$projectname`	`ConfigurationManagement`	The project name.

Keywords	Where Used	Description
`$<Property>`	`<lang>_CG`	The value of the element property with the specified name (under `C` or `CPP_CG::CG::<metatype>`). This keyword is used by the **DescriptionTemplate** property.
`$RegTlb`	`ATL`	Specifies whether the COM server needs to register its type library. Automatically expands to TRUE/ FALSE depending upon COM ATL server includes type library.
`$RhapsodyVersion`	`CG, <lang>_CG`	The current version of Rhapsody, not including the build number. This information is printed in headers and footers of generated files.
`$rhpdirectory`	`ConfigurationManagement`	The path to the `_rpy` directory, which consists of the project repository. The repository contains all the configuration items for a project.
`$Signature`	`<lang>_CG`	The operation signature, used by the **DescriptionTemplate** property.
`$state`	Properties `CPP_CG::Framework::IsInCall` `CPP_CG::Framework::IsCompletedCall`	In the code generated by Rhapsody for checking whether an application is in a given state, this keyword is replaced by the state name.
`$target`	`<Container Types>, <lang>_CG`	The target of an operation on relations. This is generally the role name. For example, in a class with a relation called `myServer`, the role name `myServer` would replace the variable `$target` when expanding properties that involve that relation. The value `add$target:c` would become: `addMyServer()` The qualifier `:c` capitalizes the role name.
`$Target`	`<lang>_CG`	The other end of the association, used by the **DescriptionTemplate** property.
`$targetDir`	`ConfigurationManagement`	The target directory.
`$ThreadModel`	`ATL`	The value of the `ThreadingModel` property (Default = `Apartment`).
`$tlbPath`	`ATL`	The full path of the COM type library file.

Keywords	Where Used	Description
`$type`	`CG, <lang>_CG`	The name of the type. For example, if you create a type named `FOO` and set its `in` property to `"const $type&"`, the generation of an `in` argument will be as follows: `"const FOO& <argname>"`
`$Type`	`<lang>_CG`	The argument type, used by the **DescriptionTemplate** property.
`$TypeName`	`ATL`	The value of the `TypeName` property, which specifies the declaration of the class type being registered (Default = `$class`).
`$unit`	`ConfigurationManagement`	Unit of collaboration. This is the name of the file that corresponds to the configuration item (package, configuration, or diagram) on which a CM command operates. If more than one unit is provided, the command is performed repeatedly in a `for each` loop.
`$VersionIndepProgID`	`ATL`	Replaced with the value of the `VersionIndepProgID` property (Default = `$component.$class`).
`$VtblName`	`<lang>_CG`	The name of the object's virtual function table, specified by the `ReactiveVtblName` property.

The following table lists the predefined Rhapsody macros used in the framework files and makefiles.

Macro	Description
`AR`	The command to build a library.
`ARFLAGS`	The flags used to build a library.
`CP`	Environment-specific copy command.
`CPP_EXT`	Environment-specific extension for C++ implementation files (for example, `.cpp`).
`DLL_CMD`	Expands to the DLL link command that initiates the DLL link phase of a build
`DLL_FLAGS`	Expands to the switches applied to the DLL link command
`H_EXT`	Environment-specific extension for C++ implementation files (for example, .h).

Macro	Description
`INCLUDE_QUALIFIER`	The qualifier used in a given environment to designate an include file in the compiler or link switches.
`LIB_CMD`	The command to build a library.
`LIB_EXT`	Environment-specific extension for library files (for example, `.lib`).
`LIB_FLAGS`	The flags used to build a library.
`LIB_NAME`	The name of a library.
`LIB_POSTFIX`	The postfix added between the main file name and the extension. The possible values are as follows: • `sim`—Simulated time (for example, `oxfsim.lib`) • `inst`—Instrumentation (for example, `oxfinst.lib`) • `siminst`—Simulated time and instrumentation (for example, `oxfsiminst.lib`) This macro is not used for DLLs.
`LIB_PREFIX`	The prefix added to the beginning of a file name. For example, the prefix “Vx” is added to VxWorks libraries. This macro is not used for DLLs.
`LINK_CMD`	Expands to the link command that initiates the link phase of a build
`LINK_FLAGS`	Expands to the link switches applied to the link command
`OBJ_EXT`	The environment-specific extension for object files (for example, `.o` or `.obj`).
`OBJS`	The intermediate object files to be built (for example, `aombrk.obj`).
`PDB_EXT`	The environment-specific extension for PDB debug files (for example, `.pdb`).
`RM`	The environment-specific remove command for deleting files.
`RMDIR`	The environment-specific remove command for deleting directories. This is used in the clean rules when you set the property `<lang>_CG::<Environment>::`**ObjectsDirectory**.

Mapping Custom Properties to Keywords

You can define custom keywords in makefile template properties and standard operations. The property name for the custom keyword should be the same as the keyword string. For example, for the keyword `$AAA`, the property name should be `AAA`.

Define the property in a specific `Subject` and `Metaclass`, as follows:

Property Type	Subject	Metaclass
Makefile	`CG/<lang>_CG`	`Component/ Configuration/ <Environment>`
Standard operations	`CG/<lang>_CG`	The keyword context (class, relation, attribute, and so on)

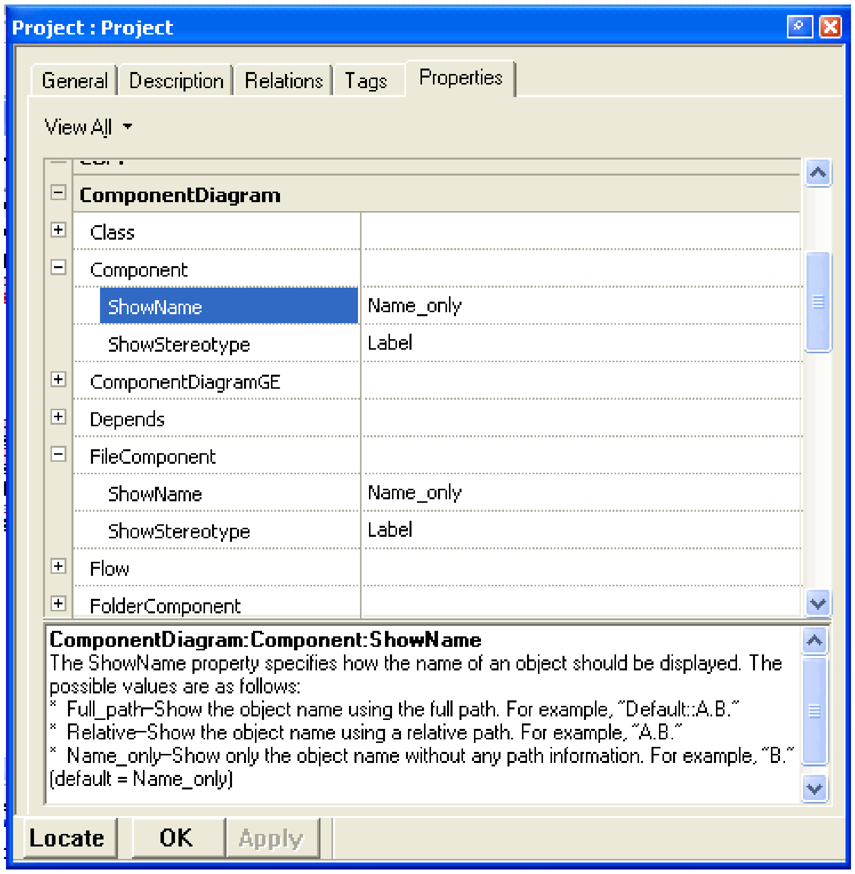
Setting Rhapsody Properties

Properties affect aspects of your model, such as the appearance of graphics in various graphic editors, how code is generated, or configuration management settings.

Using the Properties Tab in the Features Dialog Box

Rhapsody provides easy access to the properties through the interface. Either of the following methods may be used:

- With a project open, select **File > Project Properties**. The Features dialog box displays with the Properties tab already selected.
- Right-click an item in the browser and select the **Features** option from the menu. Select the **Properties** tab (shown below) to list the properties for the selected item.



ComponentDiagram	
Class	
Component	
ShowName	Name_only
ShowStereotype	Label
ComponentDiagramGE	
Depends	
FileComponent	
ShowName	Name_only
ShowStereotype	Label
Flow	
FolderComponent	

ComponentDiagram:Component: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)

The Properties tab uses a tree structure to display the subjects, metaclasses, and properties.

In the left column, the subjects are listed in boldface font; expand the plus sign to view the metaclasses for a particular subject. When you expand a metaclass, the corresponding properties are listed in the left column, with their current values listed in the right column.

For example, in the figure, the property `ObjectModelGe::Class::ShowName` can have the values `Full_path`, `Relative` (the default value), and `Name_only`.

Note that items are usually in alphabetical order; however, metaclasses that are of the same type as the context are “pushed up” to be first. For example, if you are viewing the properties of a selected class, the first metaclass displayed is `CG::Class`.

You can select the separator to resize the columns or hide (and redisplay) property names and their values. When you click on the vertical separator, a ghost line appears so you can control the column width and display.

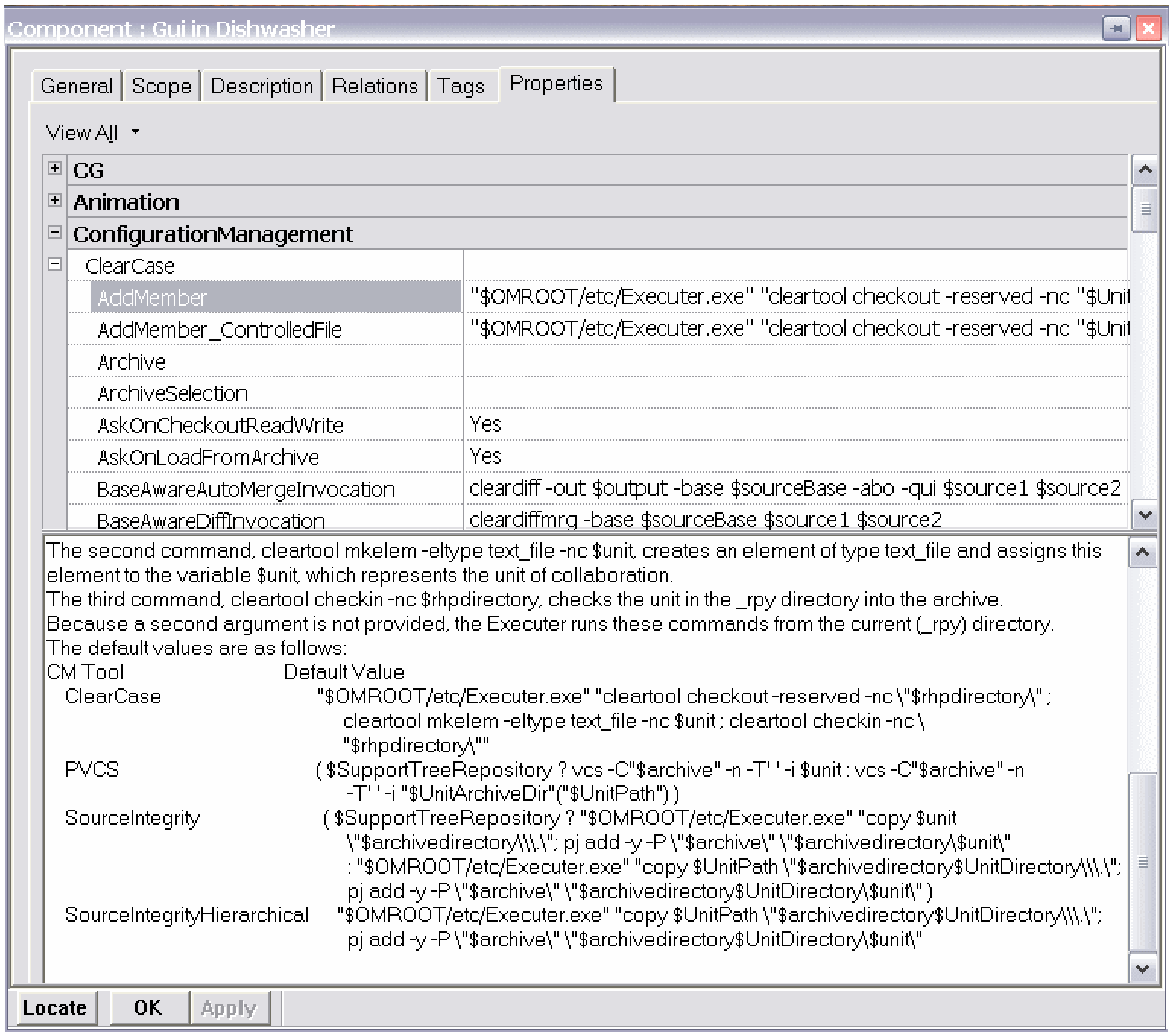
Selecting a property displays the property’s description in the bottom pane of the window. Property descriptions may carry formats such as bullets and lists.

Sizing the Features Dialog Box

Many of the properties have long tables of information. You may find it necessary to expand the size of the Features dialog box in order to see these tables without awkward text wrapping outside the table columns. The following illustration shows a long table in a widened dialog box.

Note

Any continuation of a long “Default Value”is indented under that heading until a new table entry begins.

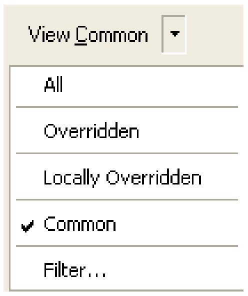


CG	
Animation	
ConfigurationManagement	
ClearCase	
AddMember	"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$Unit
AddMember_ControlledFile	"$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc "$Unit
Archive	
ArchiveSelection	
AskOnCheckoutReadWrite	Yes
AskOnLoadFromArchive	Yes
BaseAwareAutoMergeInvocation	cleardiff -out $output -base $sourceBase -abo -qui $source1 $source2
BaseAwareDiffInvocation	cleardiffmrg -base $sourceBase $source1 $source2

```
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 $rhpdirectory, checks the unit in the _rpy directory into the archive.
Because a second argument is not provided, the Executer runs these commands from the current (_rpy) directory.
The default values are as follows:
CM Tool                    Default Value
  ClearCase                    "$OMROOT/etc/Executer.exe" "cleartool checkout -reserved -nc \"$rhpdirectory\" ;
                                 cleartool mkelem -eltype text_file -nc $unit ; cleartool checkin -nc \
                                 "$rhpdirectory\""
  PVCS                         ( $SupportTreeRepository ? vcs -C"$archive" -n -T' ' -i $unit : vcs -C"$archive" -n
                                 -T' ' -i "$UnitArchiveDir"("$UnitPath") )
  SourceIntegrity              ( $SupportTreeRepository ? "$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\" )
  SourceIntegrityHierarchical   "$OMROOT/etc/Executer.exe" "copy $UnitPath \"$archivedirectory$UnitDirectory\\\.\";
                                 pj add -y -P \"$archive\" \"$archivedirectory$UnitDirectory\$unit\"
```

Filtering Views

To select the types of properties that are displayed in the Features dialog box, you can specify the view for the View pull-down menu.



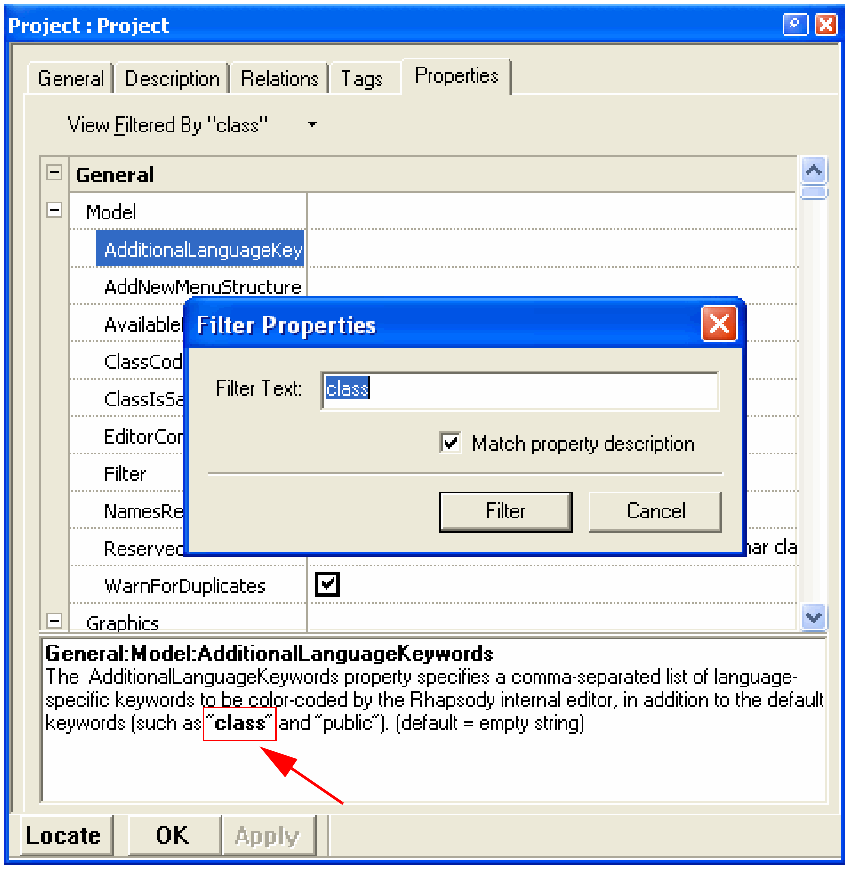
The possible views are as follows:

- **All**—Displays all the available properties, according to context.
- **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 displayed as regular, non-overridden properties. From the right-click menu, you may also select the **Un-override** option to reverse this action.
- **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.

  Note: To specify the default filter used, set the property Dialog::General::PropertiesDialogDefaultFilter.

- **Common**—Displays the properties contained in the `Dialog::<Metaclass>::CommonProperties` property. This is the default view.
- **Filter**—Displays the Filter Properties search dialog box (shown below). This feature allows the user to input and search for any text in the Property names. Selecting the “Match property description” checkbox searches property descriptions in addition to the names.

Text found in a Filter Properties search is displayed in bold type in the description area of the Features dialog box. See the figure below.



Project : Project

General	
Model	
AdditionalLanguageKey	
AddNewMenuStructure	
Available	
ClassCod	
ClassIsSa	
EditorCor	
Filter	
NamesRe	
Reserved	ar cla
WarnForDuplicates	☑
Graphics	

Filter Properties

Filter Text: class

☑ Match property description

General:Model:AdditionalLanguageKeywords
The AdditionalLanguageKeywords property specifies a comma-separated list of language-specific keywords to be color-coded by the Rhapsody internal editor, in addition to the default keywords (such as "**class**" and "public"). (default = empty string)

Changing the Common View

The common view enables you to see only a subset of the hundreds of Rhapsody properties that are available. This makes the properties GUI much easier to use. You can easily add properties that you use frequently to the common view, or remove properties that you do not use.

Adding Properties to the Common View

To add a property to the common view, do the following:

1. In the properties GUI, select the All filter so you can find the property to add to the common view.
2. Right-click the property you want to add to the common view.
3. From the pop-up menu, select **Add To common list**.

Removing Properties from the Common View

To remove a property to the common view, do the following:

1. In the properties GUI, select the Common filter so you can find the property to add to the common view.
2. Right-click the property you want to remove.
3. From the pop-up menu, select **Remove from common list**.

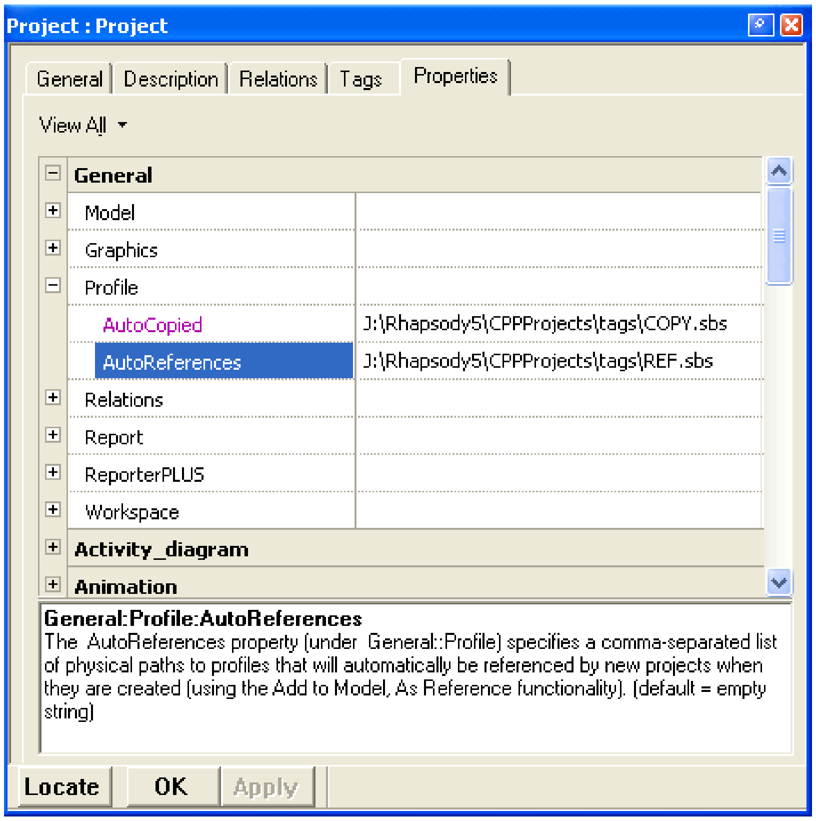
Property Controls

The Property tab uses different controls depending on the value type of the property (enum, Boolean, and so on). The following table lists the property types and the corresponding controls.

Type	Control
Boolean	Check box (a check mark = checked)
Color	Color selection box, with samples and their RGB equivalents
Enum	Drop-down list
MultiLine	Multiline edit control
Numeric value	Edit box
Text string	Text editing box

Overridden Properties

When you override the default value of a property, the property name is displayed in purple. The following figure shows an overridden property.



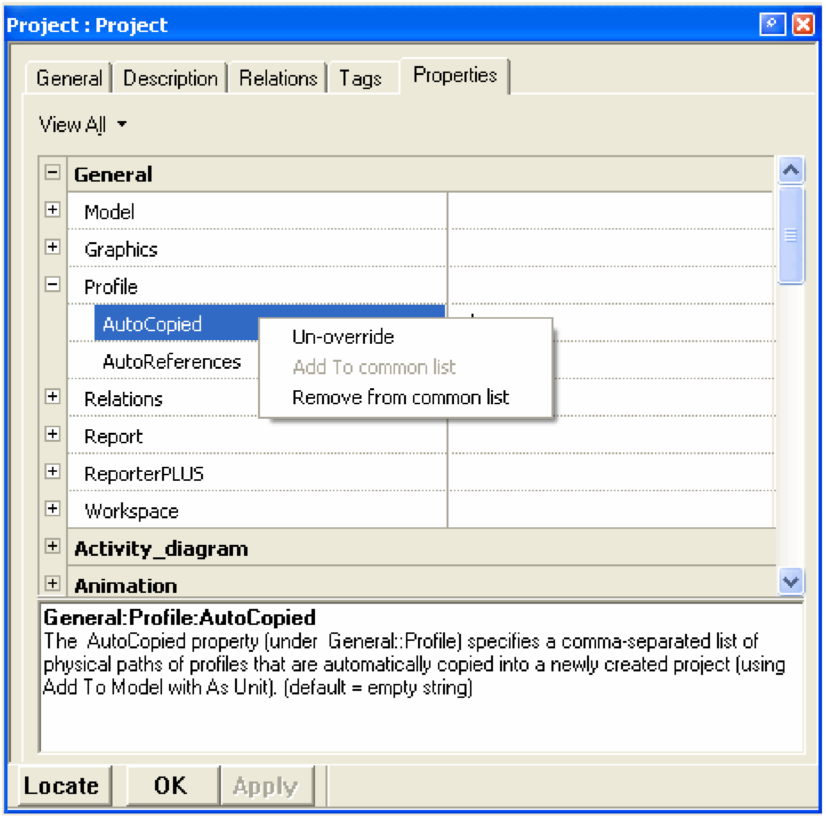
AutoCopied	J:\Rhapsody5\CPPProjects\tags\COPY.sbs
AutoReferences	J:\Rhapsody5\CPPProjects\tags\REF.sbs

General:Profile:AutoReferences
The AutoReferences property (under General::Profile) 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)

To remove an override, do the following:

1. Right-click the property value to display the Un-override command.
2. Click **Un-override**. The property is reset to the default value.
3. Click **OK to** close the dialog box.

The following figure shows the **Un-override** command.



General:Profile:AutoCopied
The AutoCopied property (under General::Profile) 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)

Changing a Property Value

To change the value of a property using the Features dialog box, do the following:

1. Select **File > Project Properties** to set properties at the project level.

   or

   To set properties at the component level, highlight the component whose property you want to change, right-click, select **Features > Properties** from the pop-up menu, then select the Properties tab.

2. If desired, select a different group of properties using the **View** pull-down menu.

3. Locate the appropriate property under the subject and metaclass.

   For example, to change the class code editor for your model, expand the `General` node in the list of subjects, then expand the `Model` metaclass to locate the `ClassCodeEditor` property.

4. Select the new value for the property in the right-hand column (for example, to change the value of the `ClassCodeEditor` property from `Internal` to `CommandLine`).

   The overridden property is displayed in purple.

5. Click **OK** to close the dialog box.

Visibility of Properties

In general, a subject is displayed for an element if it contains a metaclass that matches the metaclass of the element. The following table lists the exceptions to this rule.

Subjects Visible Only Under the Project	Subjects Visible Under Diagrams and the Project	Subjects Visible Under Only the Configuration/Component and the Project
• `General` • `RTInterface` • `RoseInterface` • `Browse` • `Report` • `IntelliVisor`	• `Diagrams` • `Statechart` • `ObjectModelGe` • `SequenceDiagram` • `UseCaseGe` • `ComponentDiagram` • `DeploymentDiagram` • `Collaboration_Diagram` • `Activity_diagram`	• `ConfigurationManagement` • `ReverseEngineering` • `CPP_ReverseEngineering`

Using the PRP Files

Default properties are assigned in the factory and site default files, `factory.prp` and `site.prp`, respectively. These files are located in the `$OMROOT\Share\Properties` directory and provide a way to tune project properties on an individual or site-wide basis without recompiling Rhapsody.

Do not change the `factory.prp` file to make individual site requirements. Instead, change the `site.prp` file for an individual site. Settings in the `site.prp` file will override the settings in the `factory.prp` file. In this way, you can always return to factory default settings in case of mistakes.

Customizing Existing Properties

You can customize the existing Rhapsody subjects, metaclasses, and/or properties or create new ones. There are many reasons for creating or modifying subjects, metaclasses and/or properties. For example, you might be using an unsupported OS, compiler (configuration), and/or configuration management tool.

When creating a new subject, you can keep existing metaclasses and properties intact. For example, the subjects OMUContainers, OMContainers, and STLContainers are all different subjects which contain the same metaclasses and properties.

Likewise, when creating a new metaclass, you can keep existing subjects and properties intact. You can also create new properties under existing subjects and metaclasses. For example, if you were using a testing tool that Rhapsody did not support, you might create new properties under an existing metaclass.

Creating a New Metaclass and Properties

You can create new metaclasses and properties using existing Rhapsody properties. For example, to add a new configuration management tool to Rhapsody, do the following:

1. In the `factory.prp` file, locate the CM tool property.
2. To the comma-separated enum values string, add the name of the new CM tool.
3. If you want this tool to be the default CM tool, change the second quoted string from “`None`” to the name of the new tool.

   When you restart Rhapsody, you will see the name of the new CM tool listed in the drop-down list of the Modify dialog box for the CM tool property.
4. Block and copy the section of code for an existing metaclass. Be sure to include the closing “`end`” for the metaclass block.
5. Rename the new metaclass to the name you specified in Step 2.
6. Edit the value of every property in the new metaclass, depending on the requirements for CM commands within the individual CM tool.

   To do the final step, refer to the documentation for the CM tool to determine the syntax for commands in that tool. Once you know what information the CM tool requires and the syntax of commands in that tool, you can use regular expression syntax and Rhapsody-internal variables to create the appropriate command strings for the tool.

Note

Do not change the original settings in the `factory.prp` file because you would not be able to roll back to the default settings.

Adding Customized Properties

You can add your own properties to existing metaclasses. You can add properties for special annotations, specification numbers, part numbers, traceability information, and any kind of comment. For example, you might require that each class be assigned a safety property and a serial number.

To add a custom property, do the following:

1. Open the `site.prp` file in the `Properties` directory.
2. Under the appropriate subject and metaclass, add the new property. Make sure to put in the correct number of `end` statements.

Adding Comments to the Properties Files

Although Rhapsody does not have a formal way to add comments to the property files, you can add comments by creating your own properties.

Do the following:

1. Create new subjects, for example:

Subject	.PRP File
Subject `SiteComment`	`site.prp`
Subject `SiteCPPComment`	`siteC++.prp`
Subject `SiteCComment`	`siteC.prp`

2. Create a new metaclass named `Comments` under each subject.
3. To each metaclass, add a new property of type `String` or `MultiLine` that contains the comment text.

If you place this information on top of your `site<Lang>.prp`, you benefit in the following ways:

- You can add comments in the file header to document why you made changes.
- Access from inside Rhapsody via the Property tab to get an overview of the version and changes inside your site properties files. However, you must keep the comments and content in sync manually.
- Gain the ability to bring site settings into the Reporter documentation.

Note

Do not use the `String` comment (`""`) inside the assigned strings of the comment properties.

Example

The following example shows a portion of the `SiteC++.prp` file with comment properties.

```
Subject SiteCPPComment

   Metaclass Comments

         Property RhpVersion String "v4.0.1 SiteC++ for Rhapsody
               Build 268921"

         Property ChangeAuthor String "Wile E. Coyote, Acme Co."

         Property LastChange String "09.07.2002"

         Property ChangeHistory MultiLine "Version 1.0
               09.07.2002"

         Property ChangeList MultiLine "
               List of Changed Properties
               Optimization Properties:
               * CPP_CG->Attribute->AccessorGenerate to False
               * CPP_CG->Attribute->MutatorGenerate to False
               * CPP_CG->Relation->RemoveKeyGenerate to False
               * CPP_CG->Relation->RemoveKeyHelpersGenerate to
               False
               Other properties:
               * None
         "

         Property GeneralComment MultiLine "
               Purpose of the changes in siteC++.prp:
               I like challenges!
               Any questions?
         "
   end
end
end
```

Including PRP Files

To include one `.prp` file in another, use the `Include` directive. Rhapsody will replace the directive with the contents of the specified file.

The syntax of the directive is as follows:

```
Include "path"
```

The specified path can be relative to the file that does the include, and should include the `.prp` extension. In addition, the path can include an environment variable. For example:

```
Include "$MY_PATH\some_dir\my_file.prp"
```

To include more than one `.prp` file, simply use multiple directives. For example:

```
Include "$MY_DIR\my_file1.prp"
Include "$MY_DIR\my_file2.prp"
```

Note the following:

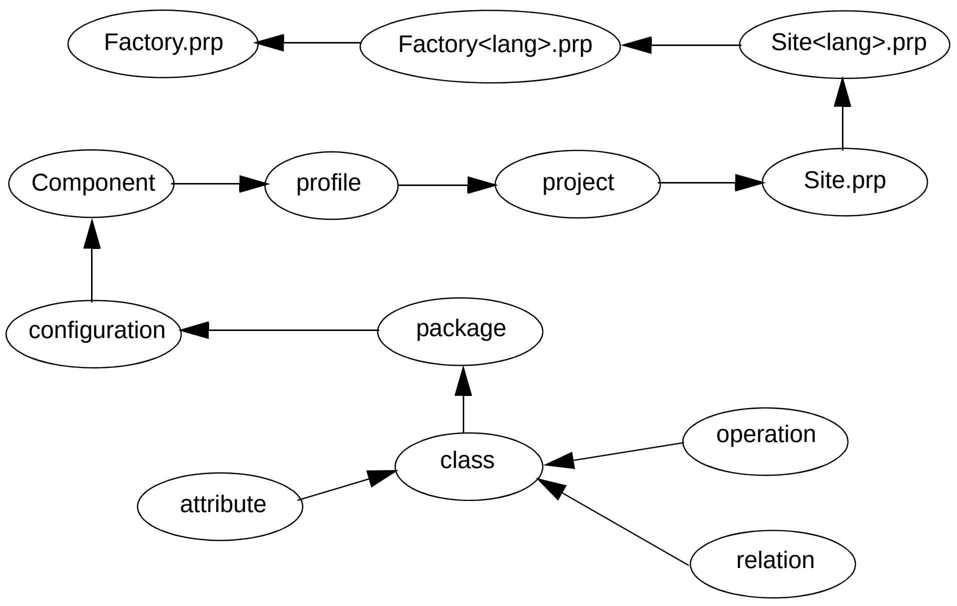
- Include statements must be outside of a Subject block—either before or after. Therefore, Rhapsody expects every included `.prp` file starts with a `Subject` line. If not, Rhapsody generates an error.
- Rhapsody does not check for loops. Therefore, a loop in the include files might cause an infinite loop when the `.prp` file is read.
- You can nest include statements. For example:

```
Include "C:\Rhapsody41\Share\Properties\IndividualSite.prp"

Subject General
   Metaclass Model
         Property BackUps Enum "None,One,Two" "Two"
   end
end

Include "..\Properties\IndividualSite.prp"

Subject General
   Metaclass Model
         Property AutoSaveInterval Int    "11"
   end
end

Include "IndividualSite.prp"
```

Property Inheritance

The level at which you set a property can affect other elements. For example, if you set a property for a dependency at the class level, and not on an individual dependency, it applies to *all* the dependencies in that class.

The following figure illustrates how property values are inherited.



Note

Note that if a stereotype is applied to an element, a property assigned to that stereotype takes precedence over the element's inherited property values (locally overridden properties take precedence over both inherited properties and those applied via a stereotype).

Concepts Used in Properties

The following sections provide a brief overview of the concepts used in the Rhapsody properties.

Static Architectures

Several properties in Rhapsody provide support for static architectures, found in hard real-time and safety-critical systems that do not use dynamic memory management during runtime. When these properties are used, all events (including timeouts and triggered events) are dynamically allocated during the initialization phase. Once allocated, the memory pool (or event queue) remains static in size during the life of the application. It is important to note that dynamic memory management capabilities are still required in order to initialize these systems. In its current implementation, Rhapsody does not generate applications that can be run in environments that are completely without dynamic memory management capabilities.

Properties that provide support for static architectures include the following:

- `BaseNumberOfInstances`
- `AdditionalNumberOfInstances`
- `ProtectStaticMemoryPool`
- `EmptyMemoryPoolCallback`
- `EmptyMemoryPoolMessage`
- `TimerMaxTimeouts`

IncludeFiles

The `IncludeFiles` property (under the `<ContainerTypes>` metaclasses) enables the selective framework includes of templates based on a particular relation implementation.

If this property is defined, includes of the files listed in the property 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.

Selective Framework Includes

Some compilers (for example, VxWorks) tend to instantiate redundant copies of templates that are defined in the C++ framework. These redundant instantiations cause the resulting code (executable) to be much larger.

To enable the use of relations without templates, a set of typeless (`void*`) containers is supplied as an alternative implementation. The generated code for relations that use the typeless containers is responsible for supplying a type-safe interface.

However, supplying typeless containers does not entirely solve the problem because templates are still included via the framework `.h` files. To resolve this issue, selective includes of framework objects must be used to avoid getting the template definitions "in the back door."

To support selective framework includes, the `oxf.h` file has been minimized to include only the most basic files. The following properties have also been added:

- `IncludeFiles`
- `ActiveIncludeFiles`
- `ReactiveIncludeFiles`
- `ProtectedIncludeFiles`
- `StaticMemoryIncludeFiles`

Reactive Classes

A class is considered reactive if it:

- Has a statechart
- Consumes events
- Is a composite

Units of Collaboration

In the property descriptions, the term "unit" refers to a unit of collaboration, which can be one of the following:

- Object type or class
- Package (`*.sbs` file)
- Configuration (`*.cfg` file)
- Object model diagram (`*.omd` file)
- Sequence diagram (`*.msc` file)

- Use case diagram (`*.ucd` file)

Instances (objects), statecharts, and events are not exchanged in isolation, but together with packages. Therefore, they are not considered units of collaboration.

The Executer

Several Rhapsody properties include calls to the Executer to execute batch files. The location of both the Executer and the target-specific batch makefile (`$makefile`) are given relative to the `$OMROOT` environment variable.

The commands that reference the Executer do so for two reasons:

- To allow definition of a single property to represent a series of commands. The Executer executes each one by calling `system()`.
- To permit execution of commands by means closely resembling those of the shell's command-line (important for wildcards and escape characters).

The Executer accepts two string arguments:

- An executable command, or list of commands separated by semicolons.
- The directory from which to run the commands. If not specified, the commands are run from the current directory. (For CM tools, the "current directory" is the `_rpy` directory).

Rhapsody Environment Variables

In addition to the properties, numerous environment variables help define the Rhapsody environment. These environment variables are stored in the `rhapsody.ini` file, normally located under `C:\Winnt` on Windows systems.

The following table lists the environment variables used by Rhapsody. For ease of use, the environment variables are listed by section in the order in which they occur in the file.

Environment Variable	Description
General section	
`OMROOT = `*`path`*	Specifies the location of the `Share` subdirectory of the Rhapsody installation. For example, if during the install you specify `D:\Rhapsody` for the destination folder, the value of `OMROOT` is as follows: `$OMROOT = D:\Rhapsody\Share`
`OMDOCROOT` = *path*	Specifies the root directory for the Rhapsody documentation set (PDF files).
`OMHELPROOT` = *path*	Specifies the root directory for the Rhapsody online help.
`RY_LICENSE_FILE`	Specifies licensing information needed by FLEX*lm*. This variable is set to one of the following values: • The path to the `license.dat` file • `1717@hostname`, where 1717 is the port number (any number between 1024 and 65534) and *hostname* is the name of the Rhapsody license server machine
`AnimationPortNumber=6423`	Specifies the port number used for communicating with the animation server.
`UseVBA` = *Boolean*	Specifies whether VBA macros can be used. For example: `UseVBA = CHECKED`
`EnableWebDownload` = *Boolean*	Enables or disables the **Download from Web** feature. For example: `EnableWebDownload=CHECKED`
`DefaultEdition` = *edition*	Specifies the default edition of Rhapsody to use. For example: `DefaultEdition = Developer`
`DefaultLanguage` = *language*	Specifies the default programming language for Rhapsody. For example: `DefaultLanguage = C++`

Environment Variable	Description
`ImplementBaseClasses=CHECKED`	Controls whether the Implement Base Classes dialog box is displayed in implicit requests. By default, this dialog box is displayed only when you explicitly invoke it. If you select the **Automatically show this window** check box on the dialog box, Rhapsody writes this line to the `rhapsody.ini` file. If desired, you can add this line directly to the `rhapsody.ini` file to automatically display the dialog box.
`RHAPSODY_AFFINITY =` *`number`*	Sets the affinity of the Rhapsody process. This variable is designed to address cases where Rhapsody has problems with more than one processor. For example, to run Rhapsody on a single processor, add the following line to the `rhapsody.ini` file: `RHAPSODY_AFFINITY=1` A zero value or lack of this variable disables the mechanism.
`NO_OUTPUT_WINDOW=CHECKED`	Disables the output window for reverse engineering (RE) messages to increase performance. RE messages are logged in the file `ReverseEngineering.log.`
Helpers section	
`name<#>=` *string*	Specifies the name of the helper. For example: `name1=Reverse Engineer Ada Source Files`

`command<#>` = *path to .exe*	Specifies the invocation command for the helper.
	For example:
	`command1=J:\Rhapsody5\`
	`AdaRevEng\bin\AdaRevEng.exe`

`initialDir<#>` = *path*	Specifies the initial directory for the helper.
	For example:
	`initialDir1=J:\Rhapsody5\`
	`AdaRevEng`

`isVisible<#> =` *0 or 1*	Specifies whether the helper is visible in the Tools menu. For example: `isVisible1=1`
`isMacro<#>` = *0 or 1*	Specifies whether the helper is a VBA macro. For example: `isMacro1=0`
`arguments<#>` = *string*	Specifies the command-line arguments for the helper. For example: `arguments1=`
`numberOfElements` = *number*	Specifies the number of helpers. For example: `numberOfElements=1`

Environment Variable	Description
CodeGen section	

`ExternalGenerator` = *path*	Specifies the path to the external generator (if used).
	For example:
	`ExternalGenerator=`
	`J:\Rhapsody5\Sodius\`
	`Launch_Sodius.bat`
	Note that this variable applies only to Rhapsody in Ada.

`ModelCodeAssociativityMode =` *enum*	Specifies the dynamic model-code associativity (DMCA) status. A value of `Dynamic` means that changes to the model are dynamically updated in the code, and vice versa.
	For example:
	`ModelCodeAssociativityMode=`
	`Dynamic`

Tip section	

`TimeStamp =`	Specifies the date and time you installed Rhapsody.
	For example:
	`TimeStamp=Mon Apr 21`
	`09:34:31 2003`

`FilePos = position`	Specifies the default position at which to display the Tip of the Day. For example: `FilePos=3200`
`StartUp` = *Boolean*	Specifies whether to display the Tip of the Day when you start Rhapsody. For example: `StartUp = 1`
Animation section	
`ViewCallStack` = *0 or 1*	Specifies whether the call stack should be visible in the next animation session. For example: `ViewCallStack=0`
`ViewEventQueue` = *0 or 1*	Specifies whether the event queue should be visible in the next animation session. For example: `ViewEventQueue=0`
Settings section	

`WindowPos` = *position*	Specifies the position of the Rhapsody window on your screen.
	For example:
	`WindowPos=0,2,-32000,-32000,`
	`-1,-1,25,38,926,669`

Environment Variable	Description
BarsLayout section	
`BrowserVisible` = *Boolean*	Specifies whether the browser should be visible, according to the settings from the last session. For example: `BrowserVisible=TRUE`
`FeaturesVisible` = *Boolean*	Specifies whether the Features dialog box should be visible, according to the settings from the last session. For example: `FeaturesVisible=FALSE`
`FeaturesFloating =` *Boolean*	Specifies whether the Features dialog box should be floating or docked, according to the settings from the last session. For example: `FeaturesFloating=TRUE`
`BrowserFloating =` *Boolean*	Specifies whether the browser should be floating or docked, according to the settings from the last session. For example: `BrowserFloating=FALSE`
`Bar<#>`	Groups the settings corresponding to each toolbar. For example: `[BarsLayout-Bar29]`
BarsLayout-Summary section	
`Bars` = *number*	Specifies the number of toolbars. For example: `Bars=30`
`ScreenCX` = *resolution*	Specifies the user screen resolution on the `X` scale. For example: `ScreenCX=1024`
`ScreenCY` = resolution	Specifies the user screen resolution on the `Y` scale. For example: `ScreenCY=768`
Plugin section	

`MTT<Version number>` = *path*	Specifies the path to the TestConductor DLL.
	For example:
	`MTT4.1=L:\Rhapsody\v41\`
	`TestConductor\`
	`TestConductor.dll`

Tornado section	
`DefaultTargetServerName` = *string*	Specifies the default target-server name used with Tornado.

Environment Variable	Description
RecentFilesList section	

`File<#> =` *path*	Lists the `.rpy` files that have been loaded recently. The maximum number of files listed is four.
	
	For example:
	`File1=J:\Rhapsody5\ProjectAda\NewFunc\NewFunc.rpy`
	`File2=J:\Rhapsody5\CPPProjects\NewFunc\NewStuff\NewStuff.rpy`
	`File3=J:\Rhapsody41MR2\AdaProject\Dishwasher\Dishwasher\Dishwasher.rpy`

Format Properties

Rhapsody uses properties under the Subject *Format* to determine the format used for displaying various graphical elements.

These properties do not appear on the Properties tab of the Features dialog box, but you can control these formatting features using the Format dialog which is displayed when you select the *Format...* item that appears on the context menu for graphical elements. This dialog allows you to set formatting options up to the project level.

In some cases, you may want to set formatting options across multiple projects. This can be done by overriding the value of formatting properties using the site.prp file.

The formatting properties that can be used are listed below.

- `DefaultSize` - specifies the default size to use for graphical elements of this type. You can change the default size for elements of a given type by selecting the *New Element Size* check box in the Make Default dialog box. In the value that is used for this property, the third coordinate represents the width of the graphical element, and the fourth coordinate represents the height of the element.
- `Fill.FillColor` - specifies the color to use to fill the background of the graphical element. Corresponds to the *Fill Color* selector on the Fill tab of the Format dialog.
- `Fill.Transparent_Fill` - used to specify whether or not the fill should be transparent. Corresponds to the *Transparent Pattern* check box on the Fill tab of the Format dialog.
- `Fill.BackgroundColor` - used as the color of the superimposed pattern if you have chosen a pattern to use for the fill. Corresponds to the *Pattern Color* selector on the Fill tab of the Format dialog.
- `Fill.FillStyle` and `Fill.FillHatch` - represent the fill pattern to use. Correspond to the *Pattern* list on the Fill tab of the Format dialog.
- `Font.Font` - specifies the font to use for the text on the graphical element. Corresponds to the font list on the Font tab of the Format dialog.
- `Font.FontColor` - specifies the color of the font to use for the text on the graphical element. Corresponds to the *Text Color* selector on the Font tab of the Format dialog.
- `Font.Size` - specifies the size of the font to use for the text on the graphical element. Corresponds to the font size list on the Font tab of the Format dialog.

- `Font.Underline` - specifies whether or not the text on the graphical element should be underlined. Corresponds to the *Underline* check box on the Font tab of the Format dialog.
- `Font.Strikeout` - specifies whether strikeout text should be used for the text on the graphical element. Corresponds to the *Strike-Out* check box on the Font tab of the Format dialog.
- `Font.Weight` - used for bolding of text on the graphical element. Corresponds to the bold/ italic control on the Font tab of the Format dialog.
- `Font.Italic` - used for italicizing text on the graphical element. Corresponds to the bold/ italic control on the Font tab of the Format dialog.
- `Line.LineColor` - specifies the color to use for the outline of the graphical element. Corresponds to the *Color* selector on the Line tab of the Format dialog.
- `Line.LineStyle` - specifies the style to use for the outline of the graphical element, for example, solid or dotted. Corresponds to the *Style* list on the Line tab of the Format dialog.
- `Line.LineWidth` - specifies the width of the line to use for the outline of the graphical element. Corresponds to the *Width* list on the Line tab of the Format dialog.

Makefiles

This section lists all the default makefiles for the environments supported by Rhapsody. See the **MakeFileContent** property for more information.

Note that, except for the JDK makefile, the makefile contents included in this section are for the C++ version of Rhapsody.

The supported environments are as follows:

- Borland
- JDK
- Linux
- Microsoft
- MicrosoftDLL
- MicrosoftWinCE.NET
- MontaVista
- MSStandardLibrary
- NucleusPLUS-PPC
- OsePPCDiab
- OseSfk
- PsosPPC
- PsosX86
- QNXNeutrinoCW
- QNXNeutrinoGCC
- Solaris2
- Solaris2GNU
- VxWorks

Borland

The default makefile for the `Borland` environment is as follows:

```
############# Target type (Debug/Release) ##################
############################################################
CPPCompileDebug=$OMCPPCompileDebug
CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug
LinkRelease=$OMLinkRelease

ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches
$OMConfigurationCPPCompileSwitches

######################################################################
# Root location of the compiler, linker, implib, include files, etc.
######################################################################
!ifndef BCROOT
!include "$(MAKEDIR)\BCROOT.INC"
!endif

.AUTODEPEND
IMPLIB  = Implib
BCCCFG  = BccW32.cfg
BCC32   = Bcc32 +$(BCCCFG)
TLINK32 = TLink32
TLIB    = TLib
BRC32   = Brc32
TASM32  = Tasm32
#
# IDE macros
#

INCLUDE_QUALIFIER=-I
RMDIR = rmdir
LIB_CMD= $(TLIB)
```

```
LINK_CMD=$(TLINK32) -v
LIB_FLAGS=$OMConfigurationLinkSwitches
LINK_FLAGS=$OMConfigurationLinkSwitches -Tpe -ap -c -L$(BCROOT)\LIB -x

TARGETS=all
CSM_EXT=.CSM


############################################
# Compiler configuration file
############################################
$(BCCCFG) :
   @Copy &&|
-w-
-R
-v
-vi
-WC
| $@

#######################################################
# Context 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
########################################################
```

```
.cpp.obj :
   @echo Compiling $<
   $(BCC32) -c @&&|
 $(CPPFLAGS)
| -o$@ $<

!if "$(TARGET_TYPE)" == "Executable"
LinkDebug=$(LinkDebug) /v
!ELIF "$(TARGET_TYPE)" == "Library"
LinkDebug=$(LinkDebug) /E
!ENDIF

!if "$(INSTRUMENTATION)" == "Animation"
INST_FLAGS=-D"OMANIMATOR"
INST_INCLUDES=-I"$(OMROOT)\LangCpp\aom";"$(OMROOT)\LangCpp\tom"
INST_LIBS="$(OMROOT)\LangCpp\lib\bc5aomanim$(LIB_EXT)"
OXF_LIBS="$(OMROOT)\LangCpp\lib\bc5oxfinst$(LIB_EXT)"
"$(OMROOT)\LangCpp\lib\bc5omComAppl$(LIB_EXT)"
SOCK_LIB=

!ELIF "$(INSTRUMENTATION)" == "Tracing"

INST_FLAGS=-DOMTRACER
INST_INCLUDES=-I"$(OMROOT)\LangCpp\aom";"$(OMROOT)\LangCpp\tom"
INST_LIBS="$(OMROOT)\LangCpp\lib\bc5tomtrace$(LIB_EXT)"
"$(OMROOT)\LangCpp\lib\bc5aomtrace$(LIB_EXT)"
OXF_LIBS="$(OMROOT)\LangCpp\lib\bc5oxfinst$(LIB_EXT)"
"$(OMROOT)\LangCpp\lib\bc5omComAppl$(LIB_EXT)"
SOCK_LIB=

!ELIF "$(INSTRUMENTATION)" == "None"

INST_FLAGS=
INST_INCLUDES=
INST_LIBS=
OXF_LIBS="$(OMROOT)\LangCpp\lib\bc5oxf$(LIB_EXT)"
SOCK_LIB=
!else
!ERROR An invalid Instrumentation $(INSTRUMENTATION) is specified.
!ENDIF

ALL_OBJS= $(OBJS) $(ADDITIONAL_OBJS)

###################################################################
# Context generated dependencies and compilation instructions
####################################################################
$OMContextDependencies

$OMFileObjPath : $OMMainImplementationFile $(OBJS)
   @echo Compiling $<
   $(BCC32) -c @&&|
 $(ConfigurationCPPCompileSwitches)
| -o$OMFileObjPath $OMMainImplementationFile

################# Linking instructions##########################
##################################################################
$(TARGET_NAME)$(EXE_EXT): $(BCCCFG) $(ALL_OBJS) $OMFileObjPath
$OMMakefileName $OMModelLibs
   @Echo Linking $<
  @$(LINK_CMD) @&&|
  $(LINK_FLAGS) +
  $(BCROOT)\LIB\c0x32.obj+
  $OMFileObjPath+
  $(OBJS)+
  $(ADDITIONAL_OBJS)+
  $(LIBS)+
  $(INST_LIBS)+
  $(OXF_LIBS)+
  $(SOCK_LIB)+
  $(BCROOT)\LIB\import32.lib+
  $(BCROOT)\LIB\cw32mti.lib
  $(TARGET_NAME)$(EXE_EXT)
```

```
|

ALL_OBJS_IN_LIB=objsRepF.dat

$(ALL_OBJS_IN_LIB) : $(ALL_OBJS)
   @&echo +$** ^^& >> $(ALL_OBJS_IN_LIB)

$(TARGET_NAME)$(LIB_EXT) : $(BCCCFG) $(ALL_OBJS_IN_LIB) $OMMakefileName
   @Echo creating library $<
   @if exist $< del /f $<
   @$(LIB_CMD) $(LIB_FLAGS) $< @$(ALL_OBJS_IN_LIB),
   @del /f $(ALL_OBJS_IN_LIB)

clean:
   @echo Cleanup
   $OMCleanOBJS
   if exist $OMFileObjPath erase $OMFileObjPath
   if exist *$(OBJ_EXT) erase *$(OBJ_EXT)
   if exist $(BCCCFG) erase $(BCCCFG)
   if exist $(TARGET_NAME)$(CSM_EXT) erase $(TARGET_NAME)$(CSM_EXT)
   if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT)
   if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT)
   $(CLEAN_OBJ_DIR)
```

JDK

The default makefile for the JDK environment is as follows:

```
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_DIR%\anim.jar;%RHAP_JARS_DIR%\animcom.jar

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
```

Linux

The default makefile for the `Linux` environment is as follows:

```
############# 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 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) \
   $(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)
```

Microsoft

The default 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 "$(RPFrameWorkDll)" == "Checked"
ConfigurationCPPCompileSwitches=$(ConfigurationCPPCompileSwitches) /D
"FRAMEWORK_DLL"
!ENDIF

!IF "$(COM)" == "Checked"
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)" == "Checked"

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)" == "Checked"

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)

!ENDIF

SOCK_LIB=wsock32.lib


!ELSEIF "$(INSTRUMENTATION)" == "None"

INST_FLAGS=

INST_INCLUDES=

INST_LIBS=

!IF "$(RPFrameWorkDll)" == "Checked"
```

```
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 ########################
###################################################################
$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)
```

MicrosoftDLL

The default makefile for the `MicrosoftDLL` environment is as follows:

```
############# 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 "$(RPFrameWorkDll)" == "Checked"
ConfigurationCPPCompileSwitches=$(ConfigurationCPPCompileSwitches) /D
"FRAMEWORK_DLL"
```

```
!ENDIF

!if "$(COM)" == "Checked"
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)" == "Checked"
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)" == "Checked"

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)" == "Checked"

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)" == "Checked"

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)
```

MicrosoftWinCE.NET

The default makefile for the `MicrosoftWinCE.NET` environment is as follows:

```
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)"=="CHECKED"

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)
```

MontaVista

The default makefile for the `MontaVista` environment is as follows:

```
############# Target type (Debug/Release) ##################
############################################################
CPPCompileDebug=$OMCPPCompileDebug
CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug
LinkRelease=$OMLinkRelease

ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches
$OMConfigurationCPPCompileSwitches

#########################################
###### Predefined macros ################
CPU=$CPU
CC=$(CPU)-gcc
LIB_CMD=$(CPU)-ar
LIB_PREFIX=mvl
RM=rm -rf
MD=mkdir -p

INCLUDE_QUALIFIER=-I

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= $(OMROOT)/LangCpp/lib/$(LIB_PREFIX)aomanim$(CPU)$(LIB_EXT)
OXF_LIBS=$(OMROOT)/LangCpp/lib/$(LIB_PREFIX)oxfinst$(CPU)$(LIB_EXT)
$(OMROOT)/LangCpp/lib/$(LIB_PREFIX)omcomappl$(CPU)$(LIB_EXT)
SOCK_LIB=

else
ifeq ($(INSTRUMENTATION),Tracing)

INST_FLAGS=-DOMTRACER
INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom
INST_LIBS=$(OMROOT)/LangCpp/lib/$(LIB_PREFIX)tomtrace$(CPU)$(LIB_EXT)
$(OMROOT)/LangCpp/lib/$(LIB_PREFIX)aomtrace$(CPU)$(LIB_EXT)
OXF_LIBS= $(OMROOT)/LangCpp/lib/$(LIB_PREFIX)oxfinst$(CPU)$(LIB_EXT)
$(OMROOT)/LangCpp/lib/$(LIB_PREFIX)omcomappl$(CPU)$(LIB_EXT)
SOCK_LIB=

else
ifeq ($(INSTRUMENTATION),None)
```

```
INST_FLAGS=
INST_INCLUDES=
INST_LIBS=
OXF_LIBS=$(OMROOT)/LangCpp/lib/$(LIB_PREFIX)oxf$(CPU)$(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 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) \
   $(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)
```

MSStandardLibrary

The default makefile for the `MSStandardLibrary` environment is as follows:

```
############# 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)
```

NucleusPLUS-PPC

The default makefile for the `NucleusPLUS-PPC` environment is as follows:

```
############# 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)

   ]
```

OsePPCDiab

The default makefile for the `OsePPCDiab` environment is as follows:

```
############# 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) $(SOCK_LIB)
######################################################################
OBJ= .$/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

#

# Let's 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
######################################################################
```

OseSfk

The default makefile for the `OseSfk` environment is as follows:

```
############# 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) $(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

#
# Let's 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
########################################################################
```

PsosPPC

The default makefile for the `PsosPPC` environment is as follows:

```
##############################################
############## Predefined flags ##############
#-------------------------------------------------------------------*
#   Before using this makefile, you must define environment variable  *
#   PSS_ROOT to point to the pSOSystem "root" directory. You must     *
#   also modify the variable PSS_BSP in this file if your BSP is      *
#   not the default BSP.                                              *
#-------------------------------------------------------------------*
#PSS_BSP        =       $(PSS_ROOT)/bsps/ads8xx

#-------------------------------------------------------------------
# PSS_DRVOBJS is a list of drivers to add to the os. When the os and
# application are built separately PSS_DRVOBJS will be included as part
# of the os download file.
# When the os and application are built together PSS_DRVOBJS will be
# included in the system download file
# The driver configuration file drv_conf.obj needs to always be included
# in PSS_DRVOBJS.
#-------------------------------------------------------------------
PSS_DRVOBJS=drv_conf.o
#-------------------------------------------------------------------
# PSS_APPOBJS is a list of object files for the application. When the
# os and application are built separately PSS_APPOBJS will be included
# as part of the application download file.
# When the os and application are built together PSS_APPOBJS will be
# included in the system download file
#-------------------------------------------------------------------
PSS_APPOBJS=
#---------------------------------------------------------------------
# PSS_APPLOPTS can be used to add non-standard linker options.
#---------------------------------------------------------------------
```

```
PSS_APPLOPTS=
#----------------------------------------------------------------------
# PSS_APPINCS can be used to add non-standard include paths for
# the application, such as directories of drivers added in addition to
# pSOSystem. If no non-standard include paths are needed, define
# PSS_APPINCS as "."
#----------------------------------------------------------------------
PSS_APPINCS=
#----------------------------------------------------------------------
# DRV_LIB?s are the libraries that you want to link with your
# application like all the networking libraries.
#
# So set DRV_LIB1....DRV_LIB5 with what ever libraries you intend to
# link.
# DRV_LIB1=
# DRV_LIB2=
# DRV_LIB3=
# DRV_LIB4=
# DRV_LIB5=
#----------------------------------------------------------------------
DRV_LIB1=
DRV_LIB2=
DRV_LIB3=
DRV_LIB4=
DRV_LIB5=

#------------------------------------------------------------------------
# PSS_COMPLIB specifies libraries to be searched.  These are specified
# as command-line options:
#
# By default PSS_COMPLIB is set to include the appropriate libc library
# if you need any other libraries, you can add them.
#
#------------------------------------------------------------------------
PSS_COMPLIB = -ld -lc
FIX_LIB     = $(PSS_ROOT)/sys/libc/libcxxsp.a
```

```
#-----------------------------------------------------------------------
# APP_LIB is the name of the library file that will contain the
# compiled application code and Library to be linked with Application
#-----------------------------------------------------------------------
APP_LIB1 = app.a $(PSS_ROOT)/sys/libc/libcxxsp.a
APP_LIB2 =
APP_LIB3 =
APP_LIB4 =
APP_LIB5 =
#-------------------------------------------------------------------------
# These next 3 lines must come before any of the application rules.
#-------------------------------------------------------------------------
PSS_CONFIG=$(PSS_ROOT)/configs/std
include $(PSS_BSP)/bsp.mk
include $(PSS_CONFIG)/configxx.mk

drv_conf.o: drv_conf.c \
   makefile \
   sys_conf.h \
   $(PSS_ROOT)/include/bspfuncs.h \
   $(PSS_ROOT)/include/configs.h  \
   $(PSS_ROOT)/include/sysvars.h  \
   $(PSS_ROOT)/include/pna.h  \
   $(PSS_BSP)/bsp.h
   $(CC) $(COPTS) -o drv_conf.o drv_conf.c

CXX_OPTS+= -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangCpp -
I$(OMROOT)/LangCpp/oxf
CXX_OPTS+= -DpSOS $(INST_FLAGS) $(INCLUDE_PATH)

COPTS_FILE2+= -Dpsosppc

%.o: %.cpp; @echo Compiling $<
   @$(CXX) $(CXXOPTS) $<
```

```
RM= rm -rf
CP= cp
MV= mv

CPPCompileDebug=$OMCPPCompileDebug
CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug
LinkRelease=$OMLinkRelease
BuildSet=$OMBuildSet

ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches
$OMConfigurationCPPCompileSwitches

INCLUDE_QUALIFIER=-I
LIB_CMD=$(LIB)
LIB_FLAGS= rvu
LINK_CMD=$(LD)
LINK_FLAGS=$OMConfigurationLinkSwitches

############### Generated macros #################
##################################################
$OMContextMacros

OBJ_DIR=$OMObjectsDir
ifeq ($(OBJ_DIR),)
CREATE_OBJ_DIR=
CLEAN_OBJ_DIR=
else
CREATE_OBJ_DIR= if [ ! -e $(OBJ_DIR) ]; then mkdir $(OBJ_DIR) ; fi
CLEAN_OBJ_DIR=  if [ -e $(OBJ_DIR) ]; then $(RM) $(OBJ_DIR) ; fi
endif

#######################################################
############# Predefined Rules ########################

ifeq ($(INSTRUMENTATION),Animation)
```

```
INST_FLAGS=-DOMANIMATOR -DUSE_IOSTREAM

INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom

INST_LIBS= $(OMROOT)/LangCpp/lib/psppcaomanim$(LIB_EXT)

OXF_LIBS=$(OMROOT)/LangCpp/lib/psppcoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/
psppcomcomappl$(LIB_EXT)

SOCK_LIB=

PSS_COMPLIB+=-lios$(DFP) -li -lcfp -lram

else

ifeq ($(INSTRUMENTATION),Tracing)

INST_FLAGS=-DOMTRACER -DUSE_IOSTREAM

INST_INCLUDES=-I$(OMROOT)/LangCpp/aom -I$(OMROOT)/LangCpp/tom

INST_LIBS=$(OMROOT)/LangCpp/lib/psppctomtrace$(LIB_EXT) $(OMROOT)/LangCpp/
lib/psppcaomtrace$(LIB_EXT)

OXF_LIBS= $(OMROOT)/LangCpp/lib/psppcoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/
lib/psppcomcomappl$(LIB_EXT)

SOCK_LIB=

PSS_COMPLIB+=-lios$(DFP) -li -lcfp -lram

else

ifeq ($(INSTRUMENTATION),None)

INST_FLAGS=

INST_INCLUDES=

INST_LIBS=

OXF_LIBS=$(OMROOT)/LangCpp/lib/psppcoxf$(LIB_EXT)

SOCK_LIB=

else

echo 'An invalid Instrumentation $(INSTRUMENTATION) is specified.'

exit

endif

endif

endif




################## Generated dependencies ########################

##################################################################

$OMContextDependencies
```

```
$OMFileObjPath : $OMMainImplementationFile $(OBJS) $(INST_LIBS) $(OXF_LIBS)
   @echo Compiling $OMMainImplementationFile
   @$(CXX) $(CXXOPTS) $(ConfigurationCPPCompileSwitches)
$OMMainImplementationFile -o $OMFileObjPath


ram$(EXE_EXT): app.a $(TARGET_NAME)$(LIB_EXT)


$(TARGET_NAME)$(EXE_EXT): ram$(EXE_EXT) $OMModelLibs
   @echo Attention
   @echo Attention ram$(EXE_EXT) was generated instead of
$(TARGET_NAME)$(EXE_EXT)
   @echo Attention


$(TARGET_NAME)$(LIB_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName
$OMFileObjPath
   @echo Building library $@
   @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS)
$(ADDITIONAL_OBJS)


clean:
   @echo Cleanup
   @$(RM) $(TARGET_NAME)$(LIB_EXT)
   @$(RM) $(TARGET_NAME)$(EXE_EXT)
   $OMCleanOBJS
   @$(RM) $OMFileObjPath
   @$(RM) *$(OBJ_EXT)
   @$(RM) ram.coff *.cfe *.cof *.elf ram.* rom.*
   @$(RM) *.map *.hex *.x *.opt *.L cmd.lnk app.*
   @$(CLEAN_OBJ_DIR)


PSS_APPOBJS+= root$(OBJ_EXT) $OMFileObjPath


APP_LIB1+= $(TARGET_NAME)$(LIB_EXT)


app.a: sys_conf.h $(TARGET_NAME)$(LIB_EXT) $(PSS_APPOBJS)
   $(RM)  app.a
   $(LIB) $(LIB_FLAGS) app.a $(PSS_APPOBJS)
```

```
APP_LIB5+= $(LIBS) \
 $(INST_LIBS) \
 $(OXF_LIBS) \
 $(INST_LIBS) \
 $(SOCK_LIB)

makefile:
   @touch $@

root$(CPP_EXT): $(OMROOT)/MakeTmpl/root$(CPP_EXT)
   @if [ ! -f $@ ]; then echo Copying file "$<" to $@ ; $(CP) "$<" $@ ; fi
   @if [ $< -nt $@ ]; then echo Warning: file "$<" is newer than file $@; fi

drv_conf.c: $(OMROOT)/MakeTmpl/ppc_drv_conf.c
   @if [ ! -f $@ ]; then echo Copying file "$<" to $@ ; $(CP) "$<" $@ ; fi
   @if [ $< -nt $@ ]; then echo Warning: file "$<" is newer than file $@; fi

sys_conf.h: $(OMROOT)/MakeTmpl/sys_conf.h
   @if [ ! -f $@ ]; then echo Copying file "$<" to $@ ; $(CP) "$<" $@ ; fi
   @if [ $< -nt $@ ]; then echo Warning: file "$<" is newer than file $@; fi
```

PsosX86

The default makefile for the `PsosX86` environment is as follows:

```
DFP=S
##############################################
############## Predefined flags ##############
#--------------------------------------------------------------------*
#   Before using this makefile, you must define environment variable  *
#   PSS_ROOT to point to the pSOSystem "root" directory. You must     *
#   also modify the variable PSS_BSP in this file if your BSP is      *
#   not the default BSP.                                              *
#--------------------------------------------------------------------*
#PSS_BSP        =       $(PSS_ROOT)/bsps/pc
#---------------------------------------------------------------------
# PSS_DRVOBJS is a list of drivers to add to the os. When the os and
# application are built separately PSS_DRVOBJS will be included as part
# of the os download file.
# When the os and application are built together PSS_DRVOBJS will be
# included in the system download file
# The driver configuration file drv_conf.obj needs to always be included
# in PSS_DRVOBJS.
#---------------------------------------------------------------------
PSS_DRVOBJS=obj/drv_conf.obj
#---------------------------------------------------------------------
# PSS_APPOBJS is a list of object files for the application. When the
# os and application are built separately PSS_APPOBJS will be included
# as part of the application download file.
# When the os and application are built together PSS_APPOBJS will be
# included in the system download file
#---------------------------------------------------------------------
PSS_APPOBJS=
#-----------------------------------------------------------------------
# PSS_APPLOPTS can be used to add non-standard linker options.
#-----------------------------------------------------------------------
```

```
PSS_APPLOPTS=-cplus -cdtorseg CODE32
#----------------------------------------------------------------------
# PSS_APPINCS can be used to add non-standard include paths for
# the application such as directories of drivers added in addition to
# pSOSystem. if no non standard include paths are needed, define
# PSS_APPINCS as "."
#----------------------------------------------------------------------
PSS_APPINCS=
#----------------------------------------------------------------------
# DRV_LIB?s are the libraries that you want to link with your
# application like all the networking libraries.
#
# So set DRV_LIB1....DRV_LIB5 with whatever libraries you intend to
# link.
# DRV_LIB1=
# DRV_LIB2=
# DRV_LIB3=
# DRV_LIB4=
# DRV_LIB5=
#----------------------------------------------------------------------
DRV_LIB1=
DRV_LIB2=
DRV_LIB3=
DRV_LIB4=
DRV_LIB5=
#----------------------------------------------------------------------
# PSS_COMPLIB can be used to include any additional library in
# the load step that you want (for example MetaWindow). Note you must
# use full path names for PSS_COMPLIB.
#----------------------------------------------------------------------
PSS_COMPLIB=    $(PSS_ROOT)/sys/libc/i386fm00.lib $(PSS_ROOT)/sys/libc/
i386fx00.lib
PSS_FIXLIB=     $(PSS_ROOT)/sys/libc/libcxxsp.lib
#----------------------------------------------------------------------
# APP_LIB is the name of the library file that will contain the
# compiled application code and Library to be linked with application
```

```
#----------------------------------------------------------------------
APP_LIB1= obj/app.lib
APP_LIB2= $(PSS_ROOT)/sys/libc/libcxxsp.lib
APP_LIB3=

APP_LIB = $(APP_LIB1) $(APP_LIB2) $(APP_LIB3)

#----------------------------------------------------------------------
# These next 5 lines must come before any of the application rules.
#----------------------------------------------------------------------
PSS_CONFIG=$(PSS_ROOT)/configs/std

include $(PSS_BSP)/bsp.mk               # board support settings
include $(PSS_BSP)/build.mk             # linker build file rules
include $(PSS_BSP)/memory.mk            # target memory layout
include $(PSS_CONFIG)/configxx.mk
#----------------------------------------------------------------------
# print out used memory layout (defined in $(PSS_BSP)/memory.mk)
#----------------------------------------------------------------------
layout:
   @echo 'Memory layout settings:'
   @echo 'OS-Code  :  $(OS_CODE_START)-$(OS_CODE_END)'
   @echo 'OS-Data  :  $(OS_DATA_START)-$(OS_DATA_END)'
   @echo 'APP-Code :  $(APP_CODE_START)-$(APP_CODE_END)'
   @echo 'APP-Data :  $(APP_DATA_START)-$(APP_DATA_END)'

obj/drv_conf.obj:drv_conf.c\
   sys_conf.h   \
   $(PSS_BSP)/bsp.h\
   $(PSS_ROOT)/include/bspfuncs.h\
   $(PSS_ROOT)/include/configs.h\
   $(PSS_ROOT)/include/sysvars.h\
   $(PSS_ROOT)/include/pna.h
   $(CC) $(COPTS) drv_conf.c -o obj/drv_conf.obj
   $(INI) $(INIFLAG) obj/drv_conf.obj
```

```
COPTS5+= -Dpsosx86


COPTS6+= -I. -I$OMDefaultSpecificationDirectory -I$(OMROOT)/LangCpp -
I$(OMROOT)/LangCpp/oxf

COPTS6+= -DpSOS -Dpsosx86 $(INST_FLAGS) $(INCLUDE_PATH)


%.obj: %.cpp; @echo Compiling $<
   @$(CXX) $(CXXOPTS) $< -o $(shell basename $< $(CPP_EXT))$(OBJ_EXT)
   @$(INI) $(INIFLAG) $(shell basename $< $(CPP_EXT))$(OBJ_EXT)


CP= cp
MV= mv


CPPCompileDebug=$OMCPPCompileDebug
CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug
LinkRelease=$OMLinkRelease
BuildSet=$OMBuildSet
ConfigurationCPPCompileSwitches=$OMConfigurationCPPCompileSwitches


INCLUDE_QUALIFIER=-I
LIB_CMD=$(LIB)
LIB_FLAGS= -idp
LINK_CMD=$(LD)
LINK_FLAGS=$OMConfigurationLinkSwitches
############### Generated macros #################
##################################################
$OMContextMacros


OBJ_DIR=$OMObjectsDir


ifeq ($(OBJ_DIR),)
CREATE_OBJ_DIR=
CLEAN_OBJ_DIR=
else
CREATE_OBJ_DIR= if [ ! -e $(OBJ_DIR) ]; then $(MKDIR) $(OBJ_DIR) ; fi
```

```
CLEAN_OBJ_DIR=  if [ -e $(OBJ_DIR) ]; then $(RM) $(OBJ_DIR) ; fi
endif

######################################################
############ Predefined Rules ########################
$(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= $(OMROOT)/LangCpp/lib/psx86aomanim$(LIB_EXT)
OXF_LIBS=$(OMROOT)/LangCpp/lib/psx86oxfinst$(LIB_EXT) $(OMROOT)/LangCpp/lib/
psx86omcomappl$(LIB_EXT)
SOCK_LIB=  $(PSS_ROOT)/sys/libc/libios$(DFP).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/psx86tomtrace$(LIB_EXT) $(OMROOT)/LangCpp/
lib/psx86aomtrace$(LIB_EXT)
OXF_LIBS= $(OMROOT)/LangCpp/lib/psx86oxfinst$(LIB_EXT) $(OMROOT)/LangCpp/
lib/psx86omcomappl$(LIB_EXT)
SOCK_LIB=  $(PSS_ROOT)/sys/libc/libios$(DFP).lib
else
ifeq ($(INSTRUMENTATION),None)
INST_FLAGS=
INST_INCLUDES=
INST_LIBS=
OXF_LIBS=$(OMROOT)/LangCpp/lib/psx86oxf$(LIB_EXT)
SOCK_LIB=
else
echo 'An invalid Instrumentation $(INSTRUMENTATION) is specified.'
exit
endif
endif
endif
################## Generated dependencies ########################
```

```
##################################################################
$OMContextDependencies


$OMFileObjPath : $OMMainImplementationFile $(OBJS) $(INST_LIBS) $(OXF_LIBS)
   @echo Compiling $OMMainImplementationFile
   @$(CXX) $(CXXOPTS) $(ConfigurationCPPCompileSwitches)
$OMMainImplementationFile -o $OMFileObjPath


ram$(EXE_EXT): obj/app.lib $(TARGET_NAME)$(LIB_EXT)


$(TARGET_NAME)$(EXE_EXT): ram$(EXE_EXT)
   @echo Attention
   @echo Attention ram$(EXE_EXT) was generated instead of
$(TARGET_NAME)$(EXE_EXT)
   @echo Attention


$(TARGET_NAME)$(LIB_EXT): $(OBJS) $(ADDITIONAL_OBJS) $OMMakefileName
$OMFileObjPath
   @echo Building library $@
   @$(LIB_CMD) $(LIB_FLAGS) $(TARGET_NAME)$(LIB_EXT) $(OBJS)
$(ADDITIONAL_OBJS)


clean:
   @echo Cleanup
   @$(RM) $(TARGET_NAME)$(LIB_EXT)
   @$(RM) $(TARGET_NAME)$(EXE_EXT)
   $OMCleanOBJS
   @$(RM) $OMFileObjPath
   @$(RM) *$(OBJ_EXT)
   @$(RM) obj
   @$(RM) lst
   @$(RM) *.map
   @$(RM) ram.bld
   @$(RM) app.bld
   @$(RM) rom.bld
   @$(RM) romos.bld
   @$(RM) os.bld
   @$(RM) *.loc
```

```
	@$(RM) *.abs
	@$(RM) *.BD
	@$(RM) *.HX
	@$(RM) *.hex
	@$(RM) *.opt
	@$(RM) lnk.cmd
	@$(RM) ram.tmp
	@$(RM) os.tmp
	@$(RM) tmpbegin
	@$(MKDIR) obj
	@$(MKDIR) lst
	@$(CLEAN_OBJ_DIR)

PSS_APPOBJS+= root$(OBJ_EXT) $OMFileObjPath

APP_LIB1+= $(TARGET_NAME)$(LIB_EXT)

obj/app.lib:sys_conf.h $(TARGET_NAME)$(LIB_EXT) $(PSS_APPOBJS)
	$(RM)  obj/app.lib
	$(LIB) obj/app.lib $(PSS_APPOBJS)

APP_LIB3 += $(LIBS) \
 $(INST_LIBS) \
 $(OXF_LIBS) \
 $(INST_LIBS) \
 $(SOCK_LIB)

lst obj:
	@if [ ! -d $@ ] ; then $(MKDIR) $@ ; echo creating directory $@ ; fi

makefile:
	@touch $@

root$(CPP_EXT): $(OMROOT)/MakeTmpl/root$(CPP_EXT)
	@if [ ! -f $@ ]; then echo Copying file "$<" to $@ ; $(CP) "$<" $@ ; fi
	@if [ $< -nt $@ ]; then echo Warning: file "$<" is newer than file $@; fi
```

```
drv_conf.c: $(OMROOT)/MakeTmpl/x86_drv_conf.c
   @if [ ! -f $@ ]; then echo Copying file "$<" to $@ ; $(CP) "$<" $@ ; fi
   @if [ $< -nt $@ ]; then echo Warning: file "$<" is newer than file $@; fi

sys_conf.h: $(OMROOT)/MakeTmpl/sys_conf.h
   @if [ ! -f $@ ]; then echo Copying file "$<" to $@ ; $(CP) "$<" $@ ; fi
   @if [ $< -nt $@ ]; then echo Warning: file "$<" is newer than file $@; fi
```

QNXNeutrinoCW

Version 4.2 introduces a different message queue implementation for the QNX environment. Now, the default style is a proprietary-style queue.

To use POSIX-style queues, do the following:

1. In the makefile, add the flag `OM_POSIX_QUEUES` to `ADDED_CPP_FLAGS`.
2. Rebuild the OXF libraries in the framework.

The default makefile for the `QNXNeutrinoCW` environment is as follows:

```
############# Target type (Debug/Release) ##################
############################################################
CPPCompileDebug=$OMCPPCompileDebug
CPPCompileRelease=$OMCPPCompileRelease
LinkDebug=$OMLinkDebug
LinkRelease=$OMLinkRelease

ConfigurationCPPCompileSwitches=$OMReusableStatechartSwitches
$OMConfigurationCPPCompileSwitches

#########################################
###### Predefined macros ################
RM=del /F
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)$(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=$(OMROOT)/LangCpp/lib/QNXCWtomtrace$(LIB_EXT) $(OMROOT)/LangCpp/
lib/QNXCWaomtrace$(LIB_EXT)

OXF_LIBS= $(OMROOT)/LangCpp/lib/QNXCWoxfinst$(LIB_EXT) $(OMROOT)/LangCpp/
lib/QNXCWomcomappl$(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

endif


.SUFFIXES: $(CPP_EXT)


######################################################################

##################### Context dependencies and commands #############

$OMContextDependencies

$OMFileObjPath : $OMMainImplementationFile $(OBJS)

          @$(CC) $(ConfigurationCPPCompileSwitches) -o  $OMFileObjPath
$OMMainImplementationFile


#####################################################################

############## Predefined 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) \
	$(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
	if exist $OMFileObjPath $(ADDITIONAL_OBJS) erase $OMFileObjPath
$(ADDITIONAL_OBJS)
	if exist $(TARGET_NAME)$(LIB_EXT) erase $(TARGET_NAME)$(LIB_EXT)
	if exist $(TARGET_NAME)$(EXE_EXT) erase $(TARGET_NAME)$(EXE_EXT)
	$(CLEAN_OBJ_DIR)
```

QNXNeutrinoGCC

Version 4.2 introduces a different message queue implementation for the QNX environment. Now, the default style is a proprietary-style queue.

To use POSIX-style queues, do the following:

1. In the makefile, add the flag `OM_POSIX_QUEUES` to `ADDED_CPP_FLAGS`.
2. Rebuild the OXF libraries in the framework.

The default makefile for the `QNXNeutrinoGCC` environment is as follows:

```
############# 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 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) \
   $(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) $(TARGET_NAME)$(LIB_EXT)
   $(RM) $(TARGET_NAME)$(EXE_EXT)
   $(CLEAN_OBJ_DIR)
```

Solaris2

The default makefile for the `Solaris2` environment is as follows:

```
############# 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 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) \
   $(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)
```

Solaris2GNU

The default makefile for the `Solaris2GNU` environment is as follows:

```
############# 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
endif

.SUFFIXES: $(CPP_EXT)

#####################################################################
##################### Context dependencies and commands #############
$OMContextDependencies

$OMFileObjPath : $OMMainImplementationFile $(OBJS)
          @$(CC) $(ConfigurationCPPCompileSwitches) -o  $OMFileObjPath
$OMMainImplementationFile



####################################################################
############## Predefined 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) \
   $(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)
```

VxWorks

The default makefile for the `VxWorks` environment is as follows:

```
############# 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

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) -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)
```

Index