IBM WebSphere Business Integration Express for Item Synchronization

Map Development Guide

V4.3

Note!

Before using this information and the product it supports, read the information in “Notices” on page 415.

10October2003

This edition of this document applies to IBM WebSphere Business Integration Express for Item Synchronization, version 4.3, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this document, email doc-comments@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2003. All rights reserved.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

About this document . . . ix
Audience . . . ix
How to use this manual . . . ix
Related documents . . . x
Typographic conventions . . . x

New in this release. . . xi
New in release 4.3 . . . xi

Part 1. Maps . . . 1

Chapter 1. Introduction to map development . . . 3
About data mapping . . . 3
Maps: A closer look . . . 5
Tools for map development . . . 7
Overview of map development. . . . 10

Chapter 2. Creating maps. . . . 13
Overview of Map Designer Express . . . 13
Creating a map: Basic steps . . . 28
Specifying standard attribute transformations . . . 35
Saving maps . . . 47
Checking completion . . . 49
Mapping standards. . . . 50

Chapter 3. Working with maps . . . 51
Opening and closing a map . . . 51
Providing map property information . . . 54
Using map documents. . . . 56
Finding information in a map . . . 60
Finding and replacing text . . . 62
Printing a map . . . 62
Deleting objects . . . 63
Using execution order . . . 66
Importing and exporting maps from InterChange Server Express . . . 67

Chapter 4. Compiling and testing maps. . . . 69
Validating a map . . . 69
Compiling a map . . . 70
Compiling a set of maps . . . 71
Testing maps . . . 72
Doing advanced debugging . . . 79
Testing maps that contain relationships . . . 80
Debugging maps . . . 85

Chapter 5. Customizing a map . . . 87
Customizing transformation steps . . . 87
Importing Java packages to Interchange Server Express . . . 136
Using variables. . . . 140
Reusing map instances . . . 144
Handling exceptions . . . 144
Creating custom data validation levels . . . 146
Understanding map execution contexts. . . . 146

© Copyright IBM Corp. 2003

Part 2. Relationships 151

Chapter 6. Introduction to Relationships 153
What is a relationship? 153
Relationships: A closer look 159
Overview of the relationship development process 165

Chapter 7. Creating relationship definitions 167
Overview of Relationship Designer Express 167
Creating relationship definitions 173
Defining identity relationships. 174
Defining lookup relationships 176
Creating the relationship table schema 178
Copying relationship and participant definitions 178
Renaming relationship or participant definitions. 179
Specifying advanced relationship settings 179
Deleting a relationship definition 183
Optimizing a relationship 184

Chapter 8. Implementing relationships 187
Implementing a relationship 187
Using lookup relationships 188
Using simple identity relationships 191
Using composite identity relationships 202
Managing child instances 207
Setting the verb 210
Performing foreign key lookups 216
Loading and unloading relationships 221

Part 3. Mapping API Reference 225

Chapter 9. BaseDLM class 227
getDBConnection() 227
getName() 229
getRelConnection() 230
implicitDBTransactionBracketing() 231
isTraceEnabled() 231
logError(), logInfo(), logWarning() 232
raiseException() 233
releaseRelConnection() 235
trace() 236

Chapter 10. BusObj class 239
Exceptions and exception types 240
Syntax for traversing hierarchical business objects 240
copy() 241
duplicate() 242
equalKeys() 242
equals() 243
equalsShallow() 244
exists() 244
getBoolean(), getDouble(), getFloat(), getInt(), getLong(), get(), getBusObj(), getBusObjArray(), getLongText(),
getString() 245
getLocale() 247
getType() 247
getVerb() 247
isBlank() 248
isKey() 248
isNull() 249

isRequired() . . . 250
keysToString() . . . 250
set() . . . 251
setContent() . . . 252
setDefaultAttrValues() . . . 253
setKeys() . . . 253
setLocale() . . . 253
setVerb() . . . 254
setWithCreate() . . . 254
toString() . . . 255
validData() . . . 256
Deprecated methods . . . 256

Chapter 11. BusObjArray class . . . 259
addElement() . . . 260
duplicate() . . . 260
elementAt() . . . 261
equals() . . . 261
getElements() . . . 262
getLastIndex() . . . 262
max() . . . 262
maxBusObjArray() . . . 263
maxBusObjs() . . . 264
min() . . . 265
minBusObjArray() . . . 266
minBusObjs() . . . 267
removeAllElements() . . . 268
removeElement() . . . 268
removeElementAt() . . . 269
setElementAt() . . . 269
size() . . . 270
sum() . . . 270
swap() . . . 270
toString() . . . 271

Chapter 12. CwDBConnection class . . . 273
beginTransaction() . . . 273
commit() . . . 274
executePreparedSQL() . . . 275
executeSQL() . . . 276
executeStoredProcedure() . . . 278
getUpdateCount() . . . 279
hasMoreRows() . . . 279
inTransaction() . . . 280
isActive() . . . 280
nextRow() . . . 281
release() . . . 281
rollBack() . . . 282

Chapter 13. CwDBStoredProcedureParam class . . . 285
CwDBStoredProcedureParam() . . . 285
getParamType() . . . 286
getValue() . . . 287

Chapter 14. DtpConnection class . . . 289
beginTran() . . . 289
commit() . . . 290
executeSQL() . . . 291
execStoredProcedure() . . . 292
getUpdateCount() . . . 293

hasMoreRows() . . . 293
inTransaction() . . . 294
nextRow() . . . 294
rollBack() . . . 295

Chapter 15. DtpDataConversion class . . . 297
getType() . . . 297
isOKToConvert() . . . 298
toBoolean() . . . 300
toDouble() . . . 301
toFloat() . . . 301
toInteger() . . . 302
toPrimitiveBoolean() . . . 303
toPrimitiveDouble() . . . 303
toPrimitiveFloat() . . . 304
toPrimitiveInt() . . . 304
toString() . . . 305

Chapter 16. DtpDate class . . . 307
DtpDate() . . . 309
addDays() . . . 310
addWeekdays() . . . 311
addYears() . . . 312
after() . . . 313
before() . . . 314
calcDays() . . . 314
calcWeekdays() . . . 315
get12MonthNames() . . . 316
get12ShortMonthNames() . . . 316
get7DayNames() . . . 316
getCWDate() . . . 317
getDayOfMonth() . . . 317
getDayOfWeek() . . . 318
getHours() . . . 318
getIntDay() . . . 318
getIntDayOfWeek() . . . 319
getIntMilliSeconds() . . . 319
getIntMinutes() . . . 319
getIntMonth() . . . 320
getIntSeconds() . . . 320
getIntYear() . . . 320
getMSSince1970() . . . 321
getMaxDate() . . . 321
getMaxDateBO() . . . 322
getMinDate() . . . 323
getMinDateBO() . . . 325
getMinutes() . . . 326
getMonth() . . . 326
getNumericMonth() . . . 326
getSeconds() . . . 327
getShortMonth() . . . 327
getYear() . . . 328
set12MonthNames() . . . 328
set12MonthNamesToDefault() . . . 329
set12ShortMonthNames() . . . 329
set12ShortMonthNamesToDefault() . . . 329
set7DayNames() . . . 330
set7DayNamesToDefault() . . . 330
toString() . . . 330

Chapter 17. DtpMapService class . . . 333
runMap() . . . 333

Chapter 18. DtpSplitString class . . . 335
DtpSplitString() . . . 335
elementAt() . . . 336
firstElement() . . . 336
getElementCount() . . . 337
getEnumeration() . . . 338
lastElement() . . . 338
nextElement() . . . 338
prevElement() . . . 339
reset() . . . 340

Chapter 19. DtpUtils class . . . 341
padLeft() . . . 341
padRight() . . . 341
stringReplace() . . . 342
truncate() . . . 343

Chapter 20. IdentityRelationship class . . . 345
addMyChildren() . . . 345
deleteMyChildren() . . . 347
foreignKeyLookup() . . . 348
foreignKeyXref() . . . 350
maintainChildVerb() . . . 352
maintainCompositeRelationship() . . . 354
maintainSimpleIdentityRelationship() . . . 356
updateMyChildren() . . . 358

Chapter 21. MapExeContext class . . . 363
getConnName() . . . 363
getInitiator() . . . 363
getLocale() . . . 364
getOriginalRequestBO() . . . 365
setConnName() . . . 366
setInitiator() . . . 366
setLocale() . . . 367
Deprecated methods . . . 368

Chapter 22. Participant class . . . 369
Participant() . . . 369
getBusObj(), getString(), getLong(), getInt(), getDouble(),
getFloat(), getBoolean() . . . 371
getInstanceId() . . . 371
getParticipantDefinition() . . . 372
getRelationshipDefinition() . . . 372
set() . . . 373
setInstanceId() . . . 373
setParticipantDefinition() . . . 374
setRelationshipDefinition() . . . 374

Chapter 23. Relationship class . . . 377
addParticipant() . . . 378
create() . . . 380
deactivateParticipant() . . . 381
deactivateParticipantByInstance() . . . 382
deleteParticipant() . . . 383
deleteParticipantByInstance() . . . 384

getNewID() . . . 385
retrieveInstances() . . . 386
retrieveParticipants() . . . 388
updateParticipant() . . . 389
updateParticipantByInstance() . . . 389
Deprecated methods . . . 390

Chapter 24. UserStoredProcedureParam class . . . 393
UserStoredProcedureParam() . . . 393
getParamDataTypeJavaObj() . . . 394
getParamDataTypeJDBC() . . . 395
getParamIndex() . . . 395
getParamIOType() . . . 396
getParamName() . . . 397
getParamValue() . . . 397
setParamDataTypeJavaObj() . . . 398
setParamDataTypeJDBC() . . . 398
setParamIndex() . . . 399
setParamIOType() . . . 399
setParamName() . . . 400
setParamValue() . . . 400

Part 4. Appendixes . . . 401

Appendix A. Message files. . . . 403
Message location . . . 403
Format for map messages . . . 405
Maintaining the files . . . 407
Operations that use message files. . . . 407

Appendix B. Attribute properties . . . 413

Notices . . . 415
Programming interface information . . . 416
Trademarks and service marks . . . 416

Index . . . 419

About this document

The IBM(R) WebSphere(R) Business Integration Express for Item Synchronization product includes Interchange Server Express, the associated Toolset Express product, the Item Synchronization collaboration, and a set of software integration adapters. Together they provide business process integration and connectivity among leading e-business technologies and enterprise applications as well as integration with the UCCnet GLOBALregistry.

This document provides an introduction to the use of maps and relationships and describes how to use Map Designer Express and Relationship Designer Express for creating and modifying maps and relationships.

Audience

This document is for connector developers, collaboration developers, and IBM WebSphere consultants who create or modify business object definitions or maps.

How to use this manual

This manual is organized as follows.

Part I: Maps	
Chapter 1, "Introduction to map development"	Is an overview of maps and the Business Integration Express for Item Synchronization mapping tools.
Chapter 2, "Creating maps"	Provides an introduction to the use of Map Designer Express for the creation and modification of maps.
Chapter 3, "Working with maps"	Describes some advanced features of Map Designer Express that you might use after creating maps.
Chapter 4, "Compiling and testing maps"	Describes how to compile a map into its executable form and how to run a test run to verify the map's correctness.
Chapter 5, "Customizing a map"	Describes how to implement maps.
Part II: Relationships	
Chapter 6, "Introduction to Relationships"	Provides an introduction to relationships, including the kinds of relationships that Business Integration Express for Item Synchronization supports and the way the system implements a relationship.
Chapter 7, "Creating relationship definitions"	Provides an introduction to the use of Relationship Designer Express for the creation and modification of relationship definitions.
Chapter 8, "Implementing relationships"	Describes how to implement relationships.
Part III: Mapping API Reference	

© Copyright IBM Corp. 2003

Chapter 9, "BaseDLM class"; Chapter 10, "BusObj class"; Chapter 11, "BusObjArray class"; Chapter 12, "CwDBConnection class"; Chapter 13, "CwDBStoredProcedureParam class"; Chapter 14, "DtpConnection class"; Chapter 15, "DtpDataConversion class"; Chapter 17, "DtpMapService class"; Chapter 18, "DtpSplitString class"; Chapter 19, "DtpUtils class"; Chapter 20, "IdentityRelationship class"; Chapter 21, "MapExeContext class"; Chapter 22, "Participant class"; Chapter 23, "Relationship class"; Chapter 24, "UserStoredProcedureParam class"	Contain reference pages for methods of classes in the Mapping API.
Appendix A, "Message files"	
Appendix B, "Attribute properties"	

Related documents

The complete set of documentation describes the features and components common to all installations of IBM WebSphere Business Integration Express for Item Synchronization, and includes reference material on specific components.

You can download, install, and view the documentation at the IBM WebSphere Business Integration Express for Item Synchronization InfoCenter, located at:

http://www.ibm.com/websphere/wbiitemsync/express/infocenter

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, information that you type, or information that the system prints on the screen.
italic or `italic`	Indicates a variable name, title name, or new term the first time that it appears
blue outline	A blue outline, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click inside the outline to jump to the object of the reference.
ProductDir	Represents the directory where the product is installed.

New in this release

This section describes the new and changed features of IBM WebSphere Business Integration Express for Item Synchronization 4.3 and its associated tools for map and relationship development, which are covered in this document.

New in release 4.3

This is the first release of Map Designer Express and Relationship Designer Express as part of the IBM Web Sphere Business Integration Express for Item Synchronization release.

© Copyright IBM Corp. 2003

Part 1. Maps

© Copyright IBM Corp. 2003

Chapter 1. Introduction to map development

This chapter provides an overview of data mapping, introduces the tools you use to implement maps, and describes map and relationship definitions.

This chapter covers the following topics:

- "About data mapping" on page 3
- "Maps: A closer look" on page 5
- "Tools for map development" on page 7
- "Overview of map development" on page 10

About data mapping

Data mapping is the process of transforming (or mapping) data from one application-specific format to another. Mapping is central to the process of transferring information between different applications, and for providing collaborations (business processes) that are independent of specific applications. By mapping data between application-specific business objects and generic business objects, WebSphere creates the environment that allows for the use of "best of breed" applications. The WebSphere business integration system provides a modular and extensible architecture for easy maintenance of your maps.

The WebSphere map development system provides comprehensive support for mapping between business objects, including the following capabilities:

- Transforming data values from one or more attributes in a source business object to one or more attributes in a destination business object
- Establishing and maintaining relationships between data entities that are equivalent but are represented differently and cannot be directly transformed
- Enabling access to external mapping resources, such as third-party mapping products and databases for performing queries

When data mapping is set up among differing applications, an event occurrence in one application is performed in any other application to which it is mapped. An event occurrence can be when data is created, retrieved, updated, or deleted.

Mapping uses *maps* that define the transfer (or transformation) of data between the source and destination business objects. In the map development environment, data is mapped from an application-specific business object to a generic business object or from a generic business object to an application-specific business object. Table 1 lists the types of mapping required.

Table 1. Mapping requirements

Direction of business object	Source business object	Destination business object	Type of map
Connector to collaboration	Application-specific	Generic	Inbound map
Collaboration to connector	Generic	Application-specific	Outbound map

© Copyright IBM Corp. 2003

Figure 1 illustrates how mapping occurs at run time, using a fictionalized Employee Management collaboration as an example.

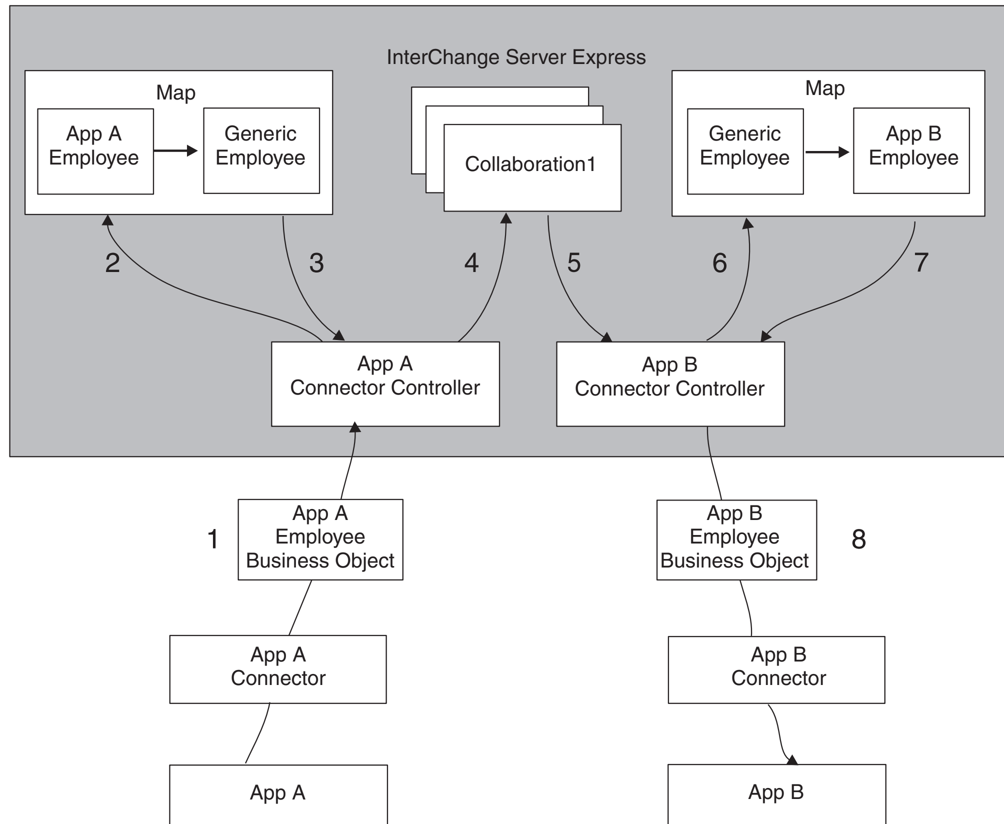


Figure 1. Data mapping at run time

The Employee Management collaboration (`Collaboration1`) receives an `Employee` business object from the source connector (`App A`), then sends an `Employee` business object to the destination connector (`App B`). Figure 1 illustrates the following sequence occurs (the numbers here correspond to the numbers in the figure):

1. An event occurs in `App A`. The `App A` connector produces an `App A Employee` business object and sends it to the `App A` connector controller.
2. The `App A` connector controller sends the `App A Employee` business object to the Employee Management collaboration (`Collaboration1`), which resides on InterChange Server Express, for mapping. The request includes the name of the data map that the server must use, based on the map name specified in the connector configuration.
3. The inbound map returns the generic `Employee` business object to the `App A` connector controller.
4. The `App A` connector controller checks the collaborations that have subscriptions to the generic `Employee` business object. In this case, `Collaboration1` has a subscription, so the connector controller hands the business object to `Collaboration1`.
5. The collaboration performs some processing, then produces another generic `Employee` business object as output, which it sends to the `App B` connector controller.
6. The `App B` connector controller sends the generic business object to InterChange Server Express, requesting mapping to the `App B Employee` business object.

7. The outbound map returns the application-specific `Employee` business object to the `App B` connector controller.
8. The `App B` connector controller passes the `App B Employee` object to the `App B` connector, which can then pass the data in the business object into `App B`.

The figure shows two types of maps in use:

- One inbound map from the `App A Employee` business object to the generic `Employee` business object used by the collaboration
- One outbound map from the generic `Employee` business object to the `App B Employee` business object

The Employee data moves in only one direction—from Application A toward Application B. If you want to exchange the Employee data in both directions between both applications, two more maps are required:

- An inbound map from the application-specific business object of Application B to the generic business object
- An outbound map from the generic business object to the application-specific business object of Application A

Maps: A closer look

As Table 2 shows, a map is a two-part entity, consisting of a map definition and a run-time object.

Map definition

You define a map to the map development system with a *map definition*. Map definitions are stored in projects in System Manager. The Map Designer Express tool provides dialogs to assist in the creation of the map definitions (often referred to simply as maps). It also handles storing the completed map definition in projects in System Manager.

For more information on how to use Map Designer Express to create map definitions, see “Creating a map: Basic steps” on page 28.

The map definition provides the following information about the map:

- The map name
- The source and destination objects of the map
- The map transformations

Map definition name

A map definition is simply a template or description of the map. It provides information on how to transform attributes of one business object to another. Therefore, the name of the map definition should identify the direction of the map and the business objects it transforms.

Source and destination business objects

Maps consist of one or more source business objects and one or more destination business objects. The *source business objects* are the ones to be transformed; the *destination business objects* are the ones that are generated with data from the source business objects.

Map transformations

The rest of the map consists of a series of transformation steps. A *transformation step* is a segment of Java code that returns the value of a destination attribute. A

map contains one transformation step for each destination attribute that is transformed. Transformations are implemented as Java code and are therefore stored in a Java source (`.java`) file.

Table 2 shows some of the transformations you can perform on a destination business object. Standard transformations include Set Value, Move, Join, Split, Submap, and Cross-Reference. You can create custom transformations with graphical function blocks.

Table 2. Transformations of a map

Transformation	Description	For more information
Standard transformations	Transformations for which Map Designer Express can autogenerate code	
Set Value	Specifying a value for a destination attribute	"Specifying a value for an attribute" on page 36
Move (Copy)	Copying a source attribute to a destination attribute	"Copying a source attribute to a destination attribute" on page 37
Join	Joining two or more source attributes into a single destination attribute	"Joining attributes" on page 38
Split	Splitting a source attribute into two or more destination attributes	"Splitting attributes" on page 40
Submap	Calling a map for a child business object	"Transforming with a submap" on page 41
Cross-Reference	Maintaining identity relationships for the business objects	"Cross-referencing identity relationships" on page 45
Custom transformations	Creating a transformation other than one of the standard transformations listed above	"Creating a Custom transformation" on page 46

When a clear correspondence exists between the source attribute and destination attribute, the transformation step simply copies the source value to the destination attribute. Other transformations can involve calculations, string manipulations, and data type conversions.

Figure 2 illustrates some typical kinds of attribute transformations:

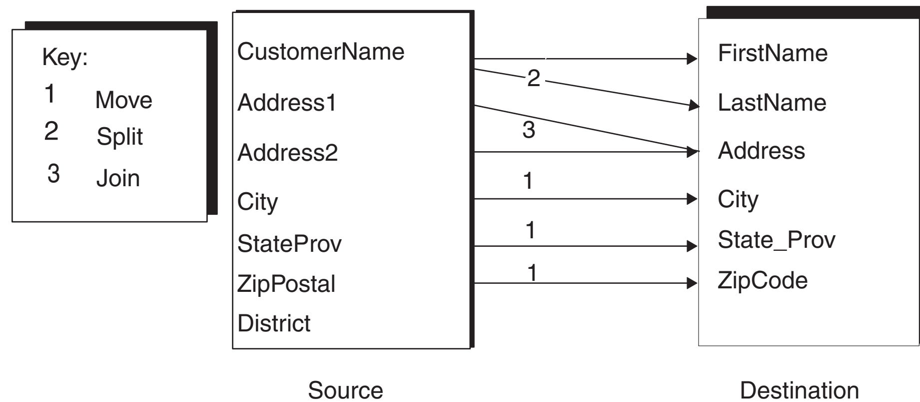


Figure 2. Typical attribute transformations

As Figure 2 shows, attributes from the source business object are typically:

- Copied to a destination attribute (`City`, `StateProv`, `ZipPostal`).

- Split into multiple destination attributes (`CustomerName`).
- Joined into one destination attribute (`Address1`, `Address2`).
- Ignored when the destination object has no equivalent attribute (`District`).

For simple transformations such as copying a value into an attribute, splitting a value into two or more attributes, or joining two or more values into one attribute, you can specify the step graphically and Map Designer Express generates the Java code. For more complex transformations, you can customize the transformation with a graphical editor.

Map instance

The map definition is a template for the run-time instantiation of the map, the *map instance*. During map execution, the Map Development system creates instances of the map based on the map definition and the transformation code.

Each map instance provides the following information:

- Basic functionality such as logging, tracing, connections, and exception handling through methods of the `BaseDLM` class
- The map execution context

  For more information, see "Understanding map execution contexts" on page 146.

Tools for map development

Table 3 shows the two graphical design tools of mapping.

Table 3. Principal components of data mapping system

Design tool	Mapping component	Description
Map Designer Express	Map	Uses Java code to specify how to transform attributes from one or more source business objects to one or more destination business objects. You typically create one map for each source business object you want to transform, though you can also break up a map into several submaps.
Relationship Designer Express	Relationship	Establishes an association between two or more data entities in the Map Development system. Relationship definitions most often associate two or more business objects. You use relationship definitions to transform data that is equivalent across business objects but is represented differently. For example, a state code for the state of Michigan might be represented as `MI` in one application and `MICH` in another. This data is equivalent but is represented differently in each application. Most maps use one, or a few, relationship definitions.

These graphical tools run on Windows 2000 and Windows XP. Therefore, these platforms are for map development.

System Manager is an additional tool that is provided for map development. It provides graphical windows to configure a map instance as well as configure a relationship object.

Map Designer Express

Map Designer Express creates and compiles maps. You can launch Map Designer Express from System Manager by selecting Map Designer Express from the Tools

menu. For other ways to launch Map Designer Express, see “Starting Map Designer Express” on page 14.. Map Designer Express provides a tab window to view map information. This window displays one of four tabs: Table tab, Diagram tab, Messages tab, or Test tab.

Figure 3 shows a map displayed in the Diagram tab of Map Designer Express.

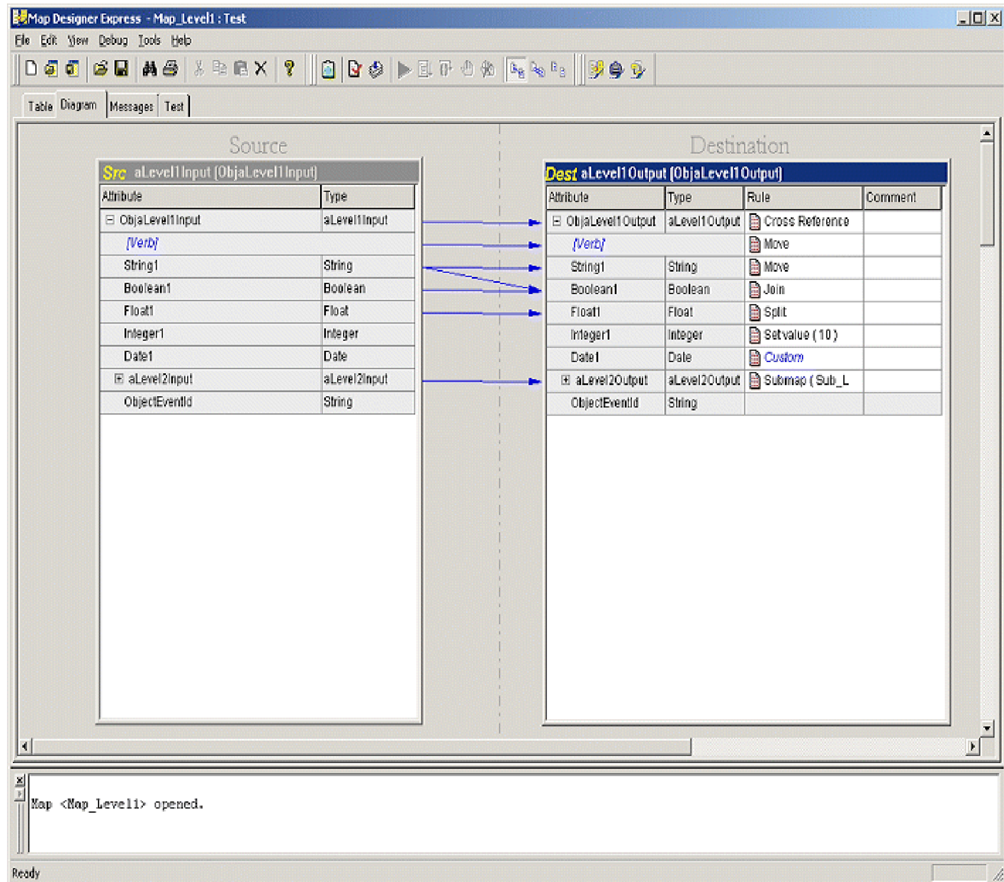


Figure 3. Map Designer Express

For information on how to use Map Designer Express to create a map, see Chapter 2, “Creating maps,” on page 13.

Relationship Designer Express

Relationship Designer Express creates relationship definitions and the table schemas that store the run-time relationship instance data. You can launch Relationship Designer Express from System Manager by selecting Relationship Designer Express from the Tools menu. Figure 4 shows several relationships

displayed in Relationship Designer Express.

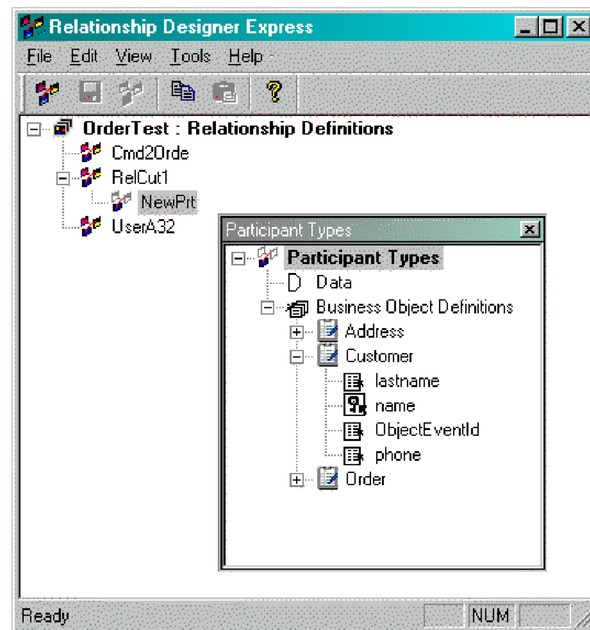


Figure 4. Relationship Designer Express

For more information on how to use Relationship Designer Express, see Chapter 7, "Creating relationship definitions," on page 167.

System Manager

System Manager is a graphical tool that provides an interface to InterChange Server Express and the repository. System Manager provides the means to manage maps and configure a map definition. You can:

- Set some general properties of a map definition, including its trace level and data validation level.
- Display the source and destination business objects of a map.
- Compile a map definition.

For more information on how to use System Manager to perform these mapping tasks, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Note: System Manager provides ways to start up Map Designer Express. For more information, see "Starting Map Designer Express" on page 14..

System Manager also provides the means to manage relationships. You can:

- Set some general properties of a relationship, including the location of its relationship tables.
- Display the participants of the relationship.

Note: System Manager also provides ways to start up Relationship Designer Express. For more information, see "Starting Relationship Designer Express" on page 167..

Overview of map development

This section provides an overview of map development, which includes the following high-level tasks:

1. Installing and setting up the map development software and installing the Java Development Kit.
2. Designing and implementing the map.

Setting up the development environment

Requirements: Before you start the development process, the following must be true:

- The map development software is installed on a machine that you can access.

  For information on how to install and start up the map development software system, see your system installation guide.
- The IBM Java Development Kit (JDK) is installed from the product CD.

  Be sure to update the `PATH` environment variable to include the installed Java directory. Restart InterChange Server Express after you have updated the path.
- System Manager is running.

  For information on starting up System Manager, see your system installation guide.
- Map Designer Express is open and connected to System Manager.

  For information on how to start Map Designer Express, see "Overview of Map Designer Express" on page 13.

Designing and implementing the map

To design and implement maps you need to do the following:

1. Learn the data formats used by all business objects involved in the map.
2. Create the map within Map Designer Express.
3. Customize any required transformation rule.
4. Define any relationships within Relationship Designer Express that the map needs.
5. Customize the mapping transformation to perform relationship management.
6. Implement error and message handling if appropriate.
7. Generate the `.java` file and compiled code. The compiled code is an executable Java class. For more information, see "Map development files" on page 11.
8. Test and debug the map, recoding as necessary.

Figure 4 provides a visual overview of map development and provides a quick reference to chapters where you can find information on specific topics.

Tip: If a team of people is available for map development, the major tasks of developing a map can be done in parallel by different members of the

development team.

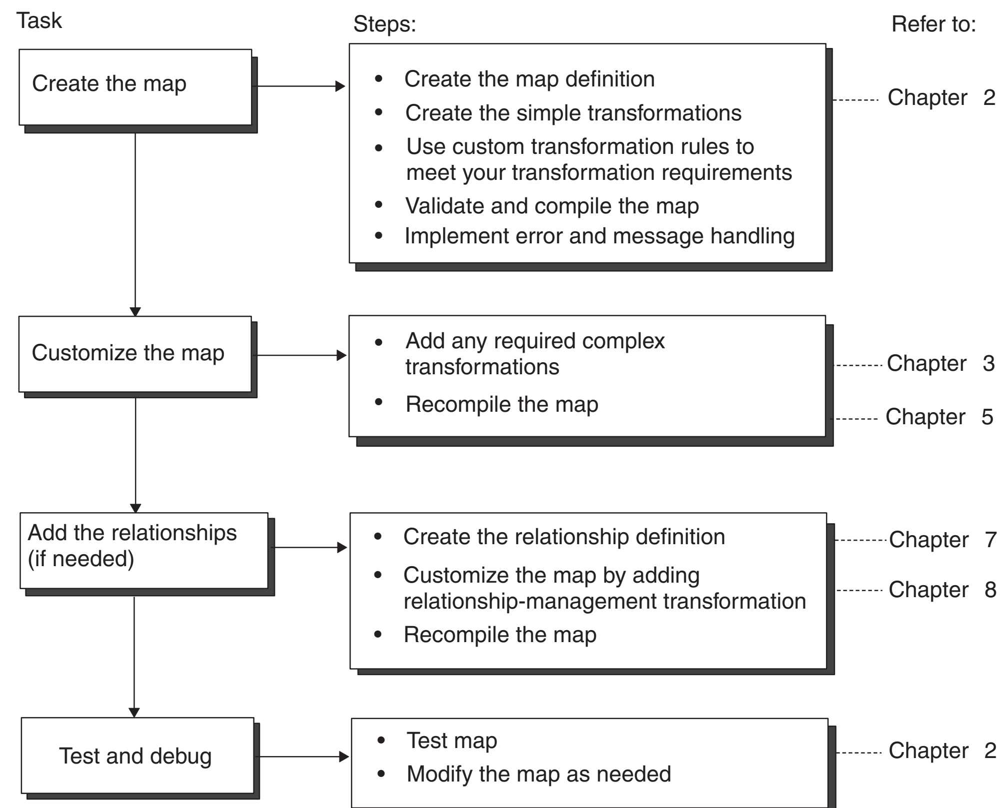


Figure 5. Overview of the map development task

Map development files

The following information forms the basis of the map:

- When you compile a map, Map Designer Express generates two types of files (`.java`, `.class`) or an optional message file (.txt) if map-specified messages are defined in the map. These files are saved in the project in System Manager.
- Map Designer Express generates a map definition when you save a map to the project in System Manager. This map definition contains general information about the map (such as map properties) as well as information about how the destination attributes are mapped.

Attention: Do not modify the *mapname*`.java` file. If you do, your changes are not reflected in the map design, which is stored in the project in System Manager. Therefore, these changes are not editable in Map Designer Express. Map Designer Express reads only the map definition.

Relationship Designer Express also stores relationship definitions in XML format in System Manager. At deployment, System Manager creates table schemas in the relationship database to contain the relationship run-time instance data. For each relationship, you can specify the location of all its relationship tables. The default location for these tables is the IBM WebSphere InterChange Server Express repository.

Table 4 lists the file types that Map Designer Express can generate (`.java`, `.class`, `.cwm`, `.bo`, `.txt`) and their locations relative to the System Manager workplace.

Table 4. Map file types

File type	Description	Location relative to System Manager workspace
.java	Generated Java code, created by Map Designer Express when you compile a map.	Stored in ProjectName\Maps\Src.
.class	Compiled Java code, created by Map Designer Express when you compile a map.	Stored in ProjectName\Maps\Classes.
.cwm	Map definition file, generated by Map Designer Express when you save a map definition.	Saved to ProjectName\Maps when "Saved" to System Manager.
.bo	Plain text file, used to save and load test run data and to save test run results.	You can save these files to any location.
.txt	Message file, created by Map Designer Express from information in the Messages tab when it compiles the map.	Stored in ProjectName\Maps\Messages.

Chapter 2. Creating maps

This chapter describes how to use the Map Designer Express to create maps.

Note: This chapter frequently uses the terms *map* and *map definition* interchangeably. When the term *map* is used, it refers to the map definition (what is accessed through Map Designer Express).

This chapter covers the following topics:

- "Overview of Map Designer Express" on page 13
- "Creating a map: Basic steps" on page 28
- "Specifying standard attribute transformations" on page 35
- "Saving maps" on page 47
- "Checking completion" on page 49
- "Mapping standards" on page 50

For background information on how the WebSphere business integration system uses maps, see Chapter 1, "Introduction to map development," on page 3.

Overview of Map Designer Express

Map Designer Express is a graphical development tool for creating and modifying maps. A *map* is made up of a series of transformation steps that define how to calculate the value for each attribute in the destination business object. Creating a map is the process of specifying the transformation steps for each destination attribute that you want to transform.

Using Map Designer Express, you can specify simple transformation steps, such as copying a source attribute to a destination attribute of the same data type, interactively using drag-and-drop. Map Designer Express automatically generates the Java code necessary to perform the transformation.

To assist with other common transformations, such as splitting a source attribute into multiple destination attributes or joining multiple source attributes into a single destination attribute, Map Designer Express prompts you for information, such as the delimiter on which to split or join, then generates the necessary Java code. To specify more complex transformations, you can define activities graphically using the Activity Editor in a custom transformation rule.

This section provides the following information as an overview to Map Designer Express:

- "Starting Map Designer Express" on page 14
- "Working in projects" on page 14
- "Layout of Map Designer Express" on page 14
- "Assigning preferences" on page 19
- "Customizing the main window" on page 21
- "Using Map Designer Express functionality" on page 23

© Copyright IBM Corp. 2003

Starting Map Designer Express

To launch Map Designer Express, you can do any of the following:

- From System Manager, you can:
  - Select Map Designer Express from the Tools menu.
  - Click a map folder in a project to enable the Map Designer Express icon in the System Manager toolbar. Then click the Map Designer Express icon.
  - Right-click the map folder in a project and select Create New Map from the Context menu.
  - Right-double-click a map to start Map Designer Express with the selected map opened.
- From a development tool, such as Business Object Designer Express, you can
  - Select Map Designer Express from the Tools menu.
  - Click the Map Designer Express icon in the Programs toolbar.
- Using a system shortcut:

```
Start-->Programs-->IBM WebSphere Business Integration Express
for Item Sync v4.3-->Toolset Express-->Development-->Map Designer Express
```

Important: For Map Designer Express to be able to access maps stored in System Manager, Map Designer Express must be connected to an instance of System Manager. The preceding steps assume that you have already started System Manager. If you have *not* started System Manager, see the *User Guide for WebSphere Business Integration Express for Item Synchronization* for more information. If System Manager is already running, Map Designer Express will automatically connect to it.

Map Designer Express displays in its own application window. You can launch more than one instance of Map Designer Express at a time to edit more than one map.

Working in projects

Map Designer Express views, edits, and modifies maps stored in System Manager on a *project* basis. A *project* is simply a logical grouping of entities for managing and deployment purposes. System Manager allows you to create multiple projects.

When Map Designer Express establishes a connection to System Manager, it obtains a list of business objects that are defined in the current project. If you add or delete a business object using Business Object Designer Express, System Manager notifies Map Designer Express, which dynamically updates the list of business object definitions.

Before you can work on a map, you select which project the map is in by entering the name of the project in the Open a Map from a Project dialog. Before you switch to another project, you need to save the maps you modified in the current project. For more information on opening a map from a project and saving a map in a project, see "Opening a map from a project in System Manager" on page 52 and "Saving a map to a project" on page 47, respectively.

Layout of Map Designer Express

When you first open Map Designer Express without specifying a map, the Map Designer Express tab window is empty and the output window does not display. When you open an existing map, the Map Designer Express window displays the Map tabs in the tab window.

Table 5 describes each of the components in the Map Designer Express main window.

Table 5. Components of the Map Designer Express window

Window area	Description	For more information
Menus	Provide options to access Map Designer Express functionality.	"Main menus of Map Designer Express" on page 23
Toolbar	Actually contains three separate toolbars, each of which provides a set of icons to access Map Designer Express functionality.	"Map Designer Express toolbars" on page 26
Map Designer Express tab window	Displays map information for an open map in one of four Map tabs.	"Table tab" on page 15 "Diagram tab" on page 17 "Messages tab" on page 18 "Test tab" on page 18
Output Window	Displays results from the compilation of a map and other status messages. If the output window is not currently displaying when Map Designer Express generates a status message, it opens this window automatically. You can clear the contents of the output window with the Clear Output option of the View menu. **Tip:** You can control whether the output window pane displays as part of the main window of Map Designer Express with the Output window option of the View menu.	N/A
Status Bar	Displays Map Designer Express status messages. **Tip:** You can control whether the status bar displays as part of the Map Designer Express window with the Status Bar option of the View menu.	N/A

The following sections describe the general layout of each of the tabs that display in Map Designer Express's tab window.

Table tab

The Table tab of Map Designer Express displays mapping information in a tabular format that lists all mapping attributes and transformations.

The Table tab consists of the following areas:

- Attribute Transformation Table
- Business Objects Pane

Attribute transformation table: The attribute transformation table presents in a tabular format all transformations associated with the map. Table 6 shows the columns that make up this table.

Table 6. Columns of the Attribute Transformation Table

Column name	Description
Exec. Order	The execution order for the destination attribute. When you add a transformation to the end of this table, Map Designer Express automatically assigns its execution order as the last in the table. You can change the execution order of an attribute by typing the desired order number in the Exec. Order field. **Note:** You can specify how Map Designer Express handles the execution order of destination attributes with the option `Defining Map: automatically adjust execution order`. By default, this option is disabled. When the option is enabled, Map Designer Express automatically adjusts the execution order of other attributes. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.
Source Attribute	The name of the source attribute for the transformation. This field provides a combo box that contains a list of all source and destination business objects with their attributes listed under them. Click the appropriate source attribute from this list. You can select multiple source attributes by clicking the Multiple Attributes entry in the combo box list. Map Designer Express displays the Multiple Attributes dialog from which you can select the attributes. **Note:** You can specify how Map Designer Express displays the source attribute name with the option `Defining Map: show full attribute path`. By default, this option is disabled and Map Designer Express displays all source attribute names as `...`*`AttrName`*. When the option is enabled, Map Designer Express displays the full attribute path: `Obj`*`SrcBusObj`*`.`*`AttrName`*. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.
Source Type	The data type of the source attribute. This field is read-only.
Destination Attribute	The name of the destination attribute for the transformation. This field provides a combo box that contains a list of all source and destination business objects with their attributes listed under them. Click the appropriate destination attribute from this list. **Note:** You can specify how Map Designer Express displays the destination attribute name with the option `Defining Map: show full attribute path`. By default, this option is disabled and Map Designer Express displays all destination attribute names as ...*`AttrName`*. When the option is enabled, Map Designer Express displays the full attribute path: `Obj`*`DestBusObj`*`.`*`AttrName`*. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.
Dest. Type	The data type of the destination attribute. This field is read-only.
Transformation Rule	The transformation rule and code for this attribute's transformation step. This field provides a combo box that contains a list of standard transformations: • None (no transformation) • Join • Move • Split • Set Value • Submap • Cross-Reference • Custom Click the appropriate transformation from this list to enter it in the field. For more information, see "Specifying standard attribute transformations" on page 35.

Table 6. Columns of the Attribute Transformation Table (continued)

Column name	Description
Comment	An informational description of the attribute's transformation. See "Setting comments in the comment field of the attribute" on page 50.

Defining a map from the Table tab: To define a map from the Table tab, follow these general steps:

1. Click in an empty cell in the Source Attribute column. From the available combo box, click the source attribute to transform.
2. Click in the corresponding cell in the Destination Attribute column. Click the destination attribute from the available combo box.
3. Click in the corresponding cell in the Transformation Rule column. This column provides a combo box:
   - For a standard transformation (Join, Move, Split, Set Value, Submap, or Cross-Reference), select the associated option from the list. Map Designer Express generates code for these standard transformations. You can customize this code as needed. For more information, see "Specifying standard attribute transformations" on page 35.
   - For a transformation that is *not* in this combo box, select Custom from the list and add the custom Java code in the Activity Editor. For more information, see "Creating a Custom transformation" on page 46.
4. Click in the corresponding cell in the Comment column. For more information, see "Setting comments in the comment field of the attribute" on page 50.

Business Objects Pane: The business objects pane presents in a list all source and destination business objects associated with the map. Its left area displays the source business objects; its right area displays the destination business objects. If the map contains a temporary business object, the business objects pane contains three areas: Source Business Object, Temporary Business Object, and Destination Business Object.

Tip: You can control whether the business objects pane displays as part of the Table tab with the Business Objects Pane option of the View menu.

Diagram tab

The Diagram tab of Map Designer Express provides a drag-and-drop interface for defining and reviewing the transformations. You view and design maps in the map workspace, which displays on the right side of the window.

The Diagram tab consists of the following areas:

- Business object browser, which displays in the project pane, on the leftmost part of the window. This browser uses a hierarchical format to list the business objects in the project in System Manager when Map Designer Express is connected to System Manager. To refresh the list of business objects in the business object browser, right-click in the business object browser and select Refresh All from the Context menu. Map Designer Express queries System Manager and updates the business object browser with the current business objects.

  Note: If you add or delete a business object from the project in System Manager, System Manager dynamically updates the list of business object definitions.

Tip: You can control whether the business object browser displays as part of the Diagram view with the Project Pane option of the View pull-down menu.

- Map workspace, which always displays the information about the current map.

  When you open a map, the map workspace displays a business object window for each source and destination business object used in the map. Each business object window lists some or all attributes defined in the business object, depending on what viewing mode is currently selected. In the case of a destination business object or temporary business object, the business object window also lists the transformation rule and comments associated with the attribute. In the map workspace, you can add, delete, or modify transformations in the map. Lines connecting attributes represent the transformations between the attributes.

  Tip: You can control which attributes display in the source and destination business objects in the Diagram tab with the options of the View--->Diagram submenu. This submenu allows you to select whether to display all attributes, only linked (mapped) attributes, or only unlinked (unmapped) attributes.

Messages tab

The Messages tab displays the map's messages. A message consists of a message ID and its associated message text.

The Messages tab is divided into two panes. The top pane is the message grid, which consists of three columns: Message ID column, Message column, and Explanation column (for comments for the entire message file). The bottom or Description pane is for entering plain text. When you enter text into the Description pane, the text is added to the top of the generated message file as comments. Map Designer Express saves any change made to the map's messages in the project of System Manager.

For more information on messages and how to use them, see Appendix A, "Message files," on page 403. For information about the format of messages, see "Format for map messages" on page 405..

When you compile a new map, Map Designer Express generates an external message file, based on the information entered in the Messages tab. This message file is saved in the message directory.

Attention: You must make all changes to a map's messages through the Messages tab of Map Designer Express. Do *not* use an external text editor to make changes to the generated message file. Any changes made from the external editor will *not* be visible to Map Designer Express because they will *not* be stored in the map definition of the project. Furthermore, such changes will be overwritten the next time you compile the map.

Test tab

The Test tab provides an interface for testing maps and viewing the results. In this tab, you can run tests to verify that transformations are working properly.

The Test tab consists of the following areas:

- Test path diagram

  The test path diagram at the top of the window shows the map test as a series of icons:

- The Source Testing Data arrow indicates the direction of the map transformation and is labeled with the business object type for the source business object that is participating in the map test.
- The Map icon represents the currently open map, which is used in the test.
- The Destination Testing Data arrow indicates the direction of the map transformation and is labeled with the business object type for the destination business object that results from the map test.

- Source Testing Data pane

  The source testing data area in the lower left window uses a hierarchical format to list the attributes of the source business object that participates in the map. Click the plus symbol (+) next to a source business object to expand it. In this area, you enter test data for the source business object.
- Destination Testing Data pane

  The destination testing data area in the lower right window uses a hierarchical format to list the attributes of the destination business object that results from the map. Click the plus symbol (+) next to a business object to expand it. In this area, you view test results data for the destination business object.

Note: Map Designer Express displays results from the test run of the map in the output window.

For more information on how to use the Test tab, see "Testing maps" on page 72.

Assigning preferences

Map Designer Express provides the Preferences dialog to allow you to customize behavior of the Map Designer Express tool. To display the Preferences dialog:
- From the View menu, select Preferences.
- Use the keyboard shortcut of `Ctrl+U`.

Figure 6 shows the Preferences dialog.

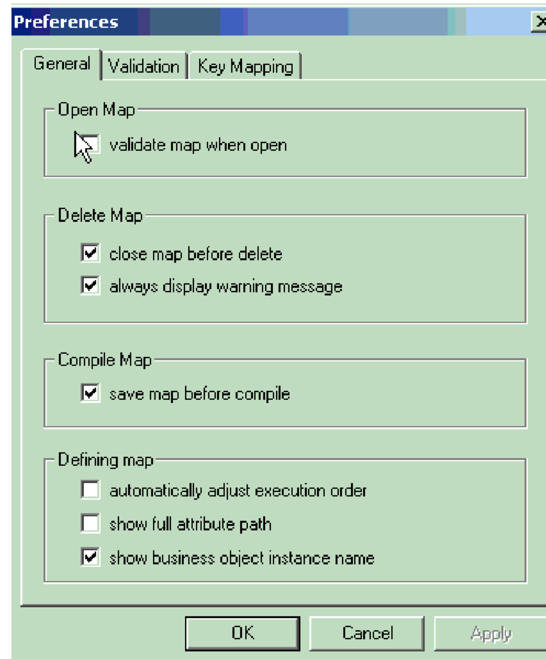


Figure 6. Preferences dialog

Map Designer Express saves preference settings in the Windows registry. Therefore, they remain in effect for the current Map Designer Express session and future sessions. The Preferences dialog provides the following tabs:

- General
- Validation
- Key Mapping

Specifying General Preferences

The General tab of the Preferences dialog displays the general preferences you can specify for how Map Designer Express manages maps.

Table 7. General Map Designer Express Preferences

General Preference	Description	For more information
Open Map		
validate map when open	When this option is enabled, Map Designer Express validates the map when it opens it. **Recommendation:** If a map uses business objects with many attributes, that is, more than a thousand attributes, enabling this option may result in the map taking a long time to open. If that is the case, and it is not desirable, you should disable this option.	"Opening a map" on page 51
Delete Map		
close map before delete	When this option is enabled, Map Designer Express always closes the currently open map before displaying the Delete Map dialog.	"Deleting maps" on page 64
always display warning message	When this option is enabled, Map Designer Express always displays a confirmation before deleting a map.	"Deleting maps" on page 64
Compile Map		
save map before compile	When this option is enabled, Map Designer Express always saves the current map to the project in System Manager before compiling it.	"Compiling a map" on page 70
Defining Map		
automatically adjust execution order	When this option is enabled, Map Designer Express automatically renumbers the execution order of destination attributes in the Table tab when execution order of an existing attribute changes.	"Using execution order" on page 66
show full attribute path	When this option is enabled, Map Designer Express shows the full attribute path for the names of source and destination attributes in the Table tab.	"Table tab" on page 15
show business object instance name	When this option is enabled, Map Designer Express displays the names of the source and destination business object *and* their variable names. When this option is disabled, Map Designer Express omits the names of the business object variables in both the Table and Diagram tabs.	"Generating business object variables" on page 140

Specifying Validation

The Validation tab of the Preferences dialog provides options you can choose for Map Designer Express to perform validations on the map when you save the map. The options are as follows:

- Show warning if verb not mapped
- Show warning if key attribute not mapped
- Show warning if required attribute not mapped
- Show warning if child business object not mapped

Map Designer Express will do the selected validation as deep as there are other transformation rules in that level.

Example: If path a.b.c is mapped, then Map Designer Express will perform these validations on business objects level a, a.b, and a.b.c.

For more information, see "Validating a map" on page 69..

Specifying Key Mappings

The Key Mapping tab of the Preferences dialog displays the key mappings for several standard transformations in the Diagram tab.

Table 8. Key Mapping Map Designer Express Preferences

Key map	Description	For more information
Move/Join/Submap	Key map to use when creating a Move, Join, or Submap transformation. Map Designer Express distinguishes between the transformations by the type and number of source attributes:	
	• Move—one source attribute that is *not* a child business object	"Copying a source attribute to a destination attribute" on page 37
	• Join—more than one source attribute that is *not* a child business object	"Joining attributes" on page 38
	• Submap—one or more source attributes that are a child business object	"Transforming with a submap" on page 41
Split	Key map to use when creating a Split transformation.	"Splitting attributes" on page 40
Cross-Reference	Key map to use for maintaining identity relationships	"Cross-referencing identity relationships" on page 45
Custom	Key map to use when creating a Custom transformation.	"Creating a Custom transformation" on page 46

The Key Mapping tab provides the following functionality:

- To change a key mapping, click in the appropriate transformation field and select the desired key map for this transformation from the combo box. Click OK.
- To return key mappings to their default values, click Use Default and then click OK.

Customizing the main window

Map Designer Express provides the following ways to customize its main window:

- "Choosing how windows display" on page 22
- "Floating a dockable window" on page 22

Choosing how windows display

When you first open Map Designer Express without specifying a map, the main window is empty with the toolbars and status bar visible. When you open a map, Map Designer Express displays the Diagram tab in the tab window and opens the output window. By default, Map Designer Express displays each of the map tabs as follows:

- Table tab—the business objects pane displays under the attribute transformation table.
- Diagram tab—the map workspace area displays and is empty.
- Messages and Test tabs—as described in "Messages tab" on page 18 and "Test tab" on page 18, respectively.

You can customize the appearance of the main window and the Map tabs with options from the View menu. Table 9 describes the options of the View pull-down menu and how they affect the appearance of the Map Designer Express window.

Table 9. View menu options for Map Designer Express window customization

View menu option	Element displayed
Toolbars	A submenu with options for each of the Map Designer Express toolbars: • Standard toolbar • Designer toolbar • Programs toolbar
Status Bar	A single-line pane in which Map Designer Express displays status information.
Business Objects Pane	A pane that displays the source and destination business objects in the Table tab of Map Designer Express.
Project Pane	A pane that displays the business object browser in the Diagram tab of Map Designer Express.
Diagram	A submenu with options for which attributes to display in the source and destination business objects in the business object windows of the Diagram tab: • All Attributes • Linked Attributes • Unlinked Attributes The Designer toolbar also provides icons for displaying these attributes.
Output Window	A small window across the bottom of the Map Designer Express window. The Clear Output option of the View menu clears all text in the output window.

Tip: When a menu option appears with a check mark to the left, the associated element displays. To turn off display of the element, select the associated menu option. The check mark disappears to indicate that the element does not currently display. Conversely, you can turn on display of an undisplayed element by selecting the associated menu option. In this case, the check mark appears beside the displaying element.

Floating a dockable window

Map Designer Express supports the following features as dockable windows:

- Toolbars in the main window:
  - Standard toolbar
  - Designer toolbar

– Programs toolbar

For more information about the features of these toolbars, see “Map Designer Express toolbars” on page 26.

- Output Window
- Find Control pane. For more information, see “Finding information in a map” on page 60.

Tip: By default, a dockable window is usually placed along the edge of the main window and moves as part of the main window. When you float a dockable window, you detach it from the main window, allowing it to function as an independent window. To float a dockable window, hold down the left mouse button, grab the border of the window and drag it onto the main window or desktop.

Using Map Designer Express functionality

You can access Map Designer Express's functionality using any of the following:

- The pull-down menus at the top of the window
- The icons in the toolbars
- Keyboard shortcuts

Main menus of Map Designer Express

Map Designer Express provides the following pull-down menus:

- File menu
- Edit menu
- View menu
- Debug menu
- Tools menu
- Help menu

The following sections describe the options of each of these menus.

Functions of the File menu: The File pull-down menu of Map Designer Express provides the options shown in Table 10.

Table 10. Options of the File menu in Map Designer Express

File menu option	Description	For more information
New...	Creates a new map file, clearing any existing map from the map workspace	“Creating a map: Basic steps” on page 28
Open (submenu)	A submenu that provides options for opening an existing map	“Opening a map” on page 51
Close	Closes the current map	“Closing a map” on page 53
Save (submenu)	A submenu that provides options for saving the current map to the same name	“Saving maps” on page 47
Save As (submenu)	A submenu that provides options for saving the current map to a name different from the map	“Saving maps” on page 47
Delete...	Deletes a specified map	“Deleting objects” on page 63
Validate Map	Validates the current map	“Validating a map” on page 69
Compile	Compiles the current map	“Compiling a map” on page 70

Table 10. Options of the File menu in Map Designer Express (continued)

File menu option	Description	For more information
Compile with Submap(s)	Compiles the current map and its submaps	"Compiling a map" on page 70
Compile All...	Compiles all or a subset of maps defined	"Compiling a set of maps" on page 71
Create Map Document...	Creates HTML files that describe the map between business objects	"Creating a map document" on page 58
View Map Document...	Displays the HTML map-document file in your HTML browser	"Viewing a map document" on page 60
Print Setup..., Print Preview, Print...	Standard Windows print options so you can preview, print, and configure a print job	"Printing a map" on page 62
Exit	Exits Map Designer Express	N/A

Functions of the Edit menu: The Edit pull-down menu of Map Designer Express provides the following options:

- Standard Windows edit options—Cut, Copy, and Paste
- Delete Current Selection—Deletes the currently selected object
- Select All—In the Diagram tab, selects all transformations between the source and destination business objects
- Insert Row—Inserts a row before the current row in the attribute transformation table of the Table tab
- Add Business Object—Displays the Add Business Object dialog to add business objects (source, destination, and temporary) to the map
- Delete Business Object—Displays the Delete Business Object dialog to delete a business object
- Find—Searches an attribute name or transformation code for text or transformation code for unmapped attributes
- Replace—searches and replaces in custom Java code or comments
- Map Properties—Displays the Map Properties window

Functions of the View menu: The View pull-down menu of Map Designer Express provides the following options:

- Business Objects Pane—When enabled, Map Designer Express displays the source and destination business objects at the bottom pane of the Table tab in the Map Designer Express window
- Diagram—A submenu that provides options for determining which attributes display in the business object windows of the Diagram tab
- Project Pane—Always enabled, Map Designer Express displays the business object browser as the left pane of the Diagram tab in the Map Designer Express window
- Clear Output—Clears the contents of the output window
- Output Window—When enabled, Map Designer Express displays status messages, including messages about opening, validating, saving, compiling, and test running the map
- Toolbars—A submenu that provides options for displaying the Map Designer Express toolbars: Standard, Designer, and Programs
- Status Bar—When enabled, Map Designer Express displays its single-line status message at the bottom of the main window

- Preferences—Displays the Preferences dialog, from which you can set Map Designer Express preferences

For information on View menu options that control display, see “Choosing how windows display” on page 22.

Functions of the Debug menu: The Debug pull-down menu provides access to the debugging facilities of Map Designer Express. It provides the following options:

- Run Test—Connects to a server and starts the test run of a map that is opened from a project
- Continue—Continues execution after it stops at a breakpoint
- Step Over—Continues execution after it stops at a breakpoint, but stops execution before executing the next attribute
- Stop Test Run--Stops the test run of a map
- Advanced--A submenu that provides options for connecting to a server for testing a map that resides in the server (Attach) and disconnecting from a server and closing a map (Detach)
- Toggle Breakpoint—Sets a breakpoint in a map, which pauses execution just before the selected attribute’s transformation
- Breakpoints—Displays all breakpoints for the map
- Clear All Breakpoints—Clears all breakpoints in the map

For more information about the use of Map Designer Express testing and debugging facilities, see “Testing maps” on page 72.

Functions of the Tools menu: The Tools pull-down menu of Map Designer Express provides options to start each of the tools:

- Map Designer Express
- Business Object Designer Express
- Relationship Designer Express

Functions of the Help Menu: Map Designer Express provides a standard Help menu with the following options:

- Help Topics
- Documentation
- About Map Designer Express

Context menu

The Context menu is a shortcut menu that is available, by right-clicking, from numerous places, such as the transformation rule column, row header in the Table view, child business object in the source testing pane, or edit box in a dialog.

A menu opens that contains useful commands, which change depending on where you click.

Example: Clicking in the transformation rule column opens a Context menu that provides the following options:

- Open—Opens the corresponding dialog box for the transformation rule, such as Join, Split, and Submap. For custom transformations, opens the Activity Editor.
- Open in New Window—For custom transformations, opens a new instance of the Activity Editor to show the detail of the transformation rule.

- View Source—Shows the transformation's corresponding Java code in the Activity Editor. The code will always be read-only.

Note: The default action when you double-click the transformation cell is Open. If Open is not available for that transformation, then a message saying that the action is not available is displayed in the status bar.

Map Designer Express toolbars

Map Designer Express provides three toolbars with common tasks you need to perform:

- Standard toolbar
- Designer toolbar
- Programs toolbar

These toolbars are dockable; that is, you can detach them from the palette of the main window and float them over the main window or the desktop.

Tip: To identify the purpose of each toolbar button, roll over each button with your mouse cursor.

Standard toolbar: Figure 7 shows the Standard toolbar.

Figure 7. Standard toolbar

The following list provides the function of each Standard toolbar button, left to right:

1. New map
2. Open
3. Save to project
4. Open from file
5. Save to file
6. Find in map
7. Print map
8. Cut
9. Copy
10. Paste
11. Delete
12. Help

Designer toolbar: Figure 8 shows the Designer toolbar.

Figure 8. Designer toolbar

The following list provides the function of each Designer toolbar button, left to right:

1. Add Business Object

2. Validate
3. Compile
4. Run Test
5. Continue
6. Step over
7. Toggle Breakpoints
8. Clear All Breakpoints
9. All Attributes
10. Linked Attributes
11. Unlinked Attributes

Programs: Figure 9 shows the Programs toolbar.

Figure 9. Programs toolbar

The following list provides the function of each Programs toolbar button, left to right:

1. Map Designer Express
2. Business Object Designer Express
3. Relationship Designer Express

Keyboard shortcuts

Map Designer Express provides the keyboard shortcuts shown in Table 11 for many of the menu options.

Table 11. Keyboard shortcuts for Map Designer Express

Keyboard shortcut	Description	For more information
`Ctrl+E`	Save the current map definition to a map definition file	"Saving a map to a file" on page 48
`Ctrl+F`	Display Find control panel to locate text or unlinked attributes in the map (use `Ctrl+H` for replace)	"Finding information in a map" on page 60
`Ctrl+H`	Display Replace dialog to find and replace text in customized Java Code and comments of transformation rules.	"Finding and replacing text" on page 62
`Ctrl+I`	Open a map definition file	"Opening a map from a file" on page 53
`Ctrl+M`	View a map document	"Viewing a map document" on page 60
`Ctrl+N`	Display the New Map wizard to create a new map	"Creating a map: Basic steps" on page 28
`Ctrl+O`	Open a map definition from the project in System Manager	"Opening a map from a project in System Manager" on page 52
`Ctrl+P`	Print the map definition	"Printing a map" on page 62
`Ctrl+S`	In Map Designer Express main window—Save the current map definition to the project in System Manager	"Saving a map to a project" on page 47
`Ctrl+U`	Display the Preferences dialog to set Map Designer Express preferences	"Assigning preferences" on page 19
`Ctrl+Alt+F`	Save the current map definition to a map definition file with a different name (Save As)	"Saving a map to a file" on page 48

Table 11. Keyboard shortcuts for Map Designer Express (continued)

Keyboard shortcut	Description	For more information
Ctrl+Alt+S	Save the current map definition to the project in System Manager with a different name (Save As)	"Saving a map to a project" on page 47
Ctrl+Shift+P	Display the Print Setup dialog to specify information for printing the map definition	"Printing a map" on page 62
Ctrl+Enter	Display the Map Properties dialog, from which you can set general and business object properties for the map	"Providing map property information" on page 54
F7	Compile the current map	"Compiling a map" on page 70
Alt+F4	Close the current map	"Closing a map" on page 53
Del	Delete the currently selected entity	N/A
F1	Display context-sensitive help for the current dialog or window	N/A
Ctrl+F7	Compile all or a subset of maps defined in System Manager	"Compiling a set of maps" on page 71
F8	During a test run, continue a paused map by executing until the end of the map or another active breakpoint	"Processing breakpoints" on page 78
F9	Toggle the state of a breakpoint for a transformation rule	"Setting breakpoints" on page 75
F10	During a test run, continue a paused map by executing the next single step	"Processing breakpoints" on page 78

Creating a map: Basic steps

Map Designer Express provides a New Map wizard to assist you in creating a map definition. Follow these basic steps to create a new map:

1. Create a new map file with the New Map wizard. Specify the project, the source and destination business objects, and the name for the new map. For help in running the New Map wizard, see "Creating the map definition" on page 29.
2. Set the verb for each destination business object. In most cases, destination business objects have the same verb as source business objects. You can also set the value of the verb always to be a specific value. For help setting the verb, see "Setting the destination business object verb" on page 35.
3. Specify the transformation steps for each destination attribute that you want to map. How you do this depends on what kind of transformation is required. For more information on specifying transformation steps, see "Specifying standard attribute transformations" on page 35.
4. Specify the comment for the destination attribute. Although this information is optional, it greatly improves readability of the map information in Map Designer Express. For more information, see "Setting comments in the comment field of the attribute" on page 50.
5. Save, validate, and compile the map. For more information on saving, see "Saving maps" on page 47. For information on compiling, see "Compiling a map" on page 70..
6. Test and debug the map. For more information on testing and debugging, see "Testing maps" on page 72..

Creating the map definition

Map Designer Express provides a New Map wizard to assist in the creation of a map definition. To create a map definition:

1. Start the New Map wizard in any of the following ways:
   - Select New from the File menu to create a new map.
   - Use the keyboard shortcut of `Ctrl+N`.
   - In the Standard toolbar, click the New Map button.

   Result: Map Designer Express displays the first window of the New Map wizard.

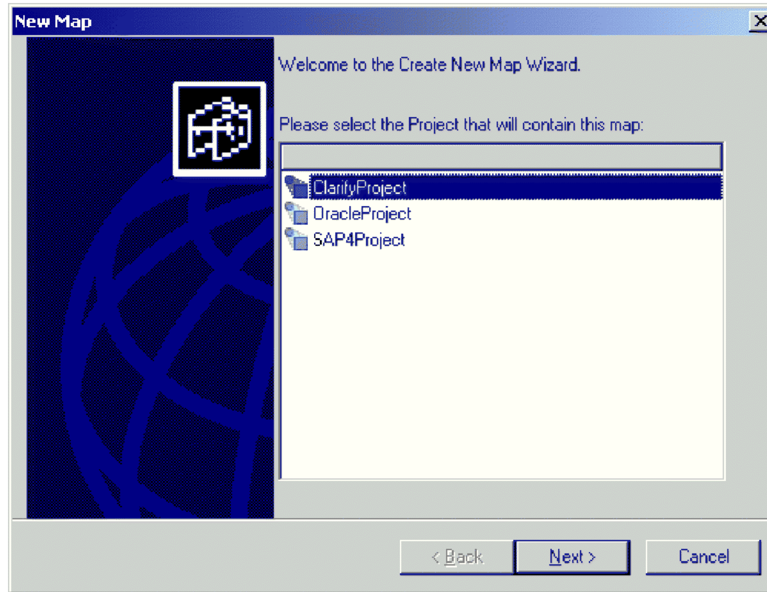


Figure 10. Welcome window of New Map wizard

2. From the list box, select the name of the project for which you want to create the map.
3. Select the business object you will use as the source business object for the map. You can select one or more source business objects by clicking in the Use

column of each desired business object. Then click Next to continue.

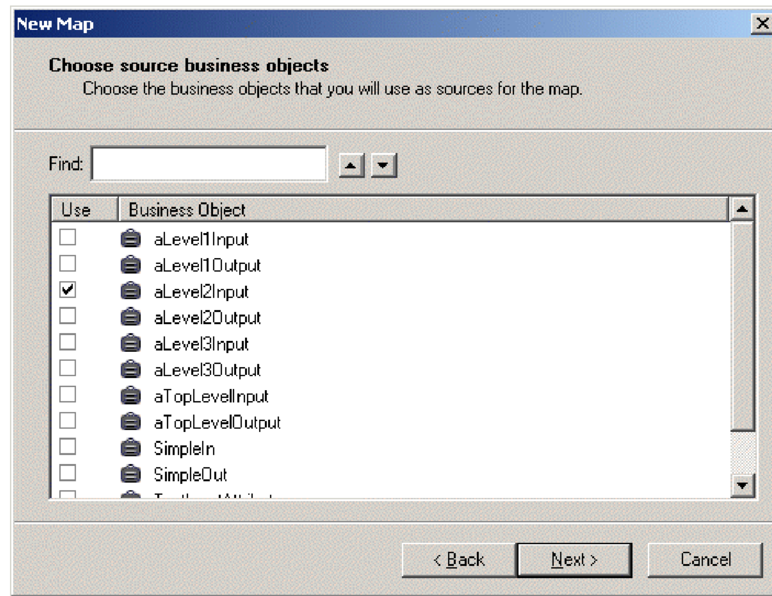


Figure 11. Selecting source business objects

Tip: To locate a particular business object, enter its name in the Find field. The up and down arrows scroll through the business object list. Click Next to continue.

The New Map wizard does *not* require that you specify the source business object. You can click Next without selecting the source business object to postpone specifying this business object definition. You can specify it at a later time in the map workspace of the Diagram tab. For more information, see "Creating the source and destination business objects" on page 32.

Note: If you add or delete a business object from System Manager, it dynamically updates the list of business object definitions.

4. Select the business object type you will use as the destination business object for the map. You can select one or more destination business objects by clicking

in the Use column of each desired business object. Then click Next to continue.

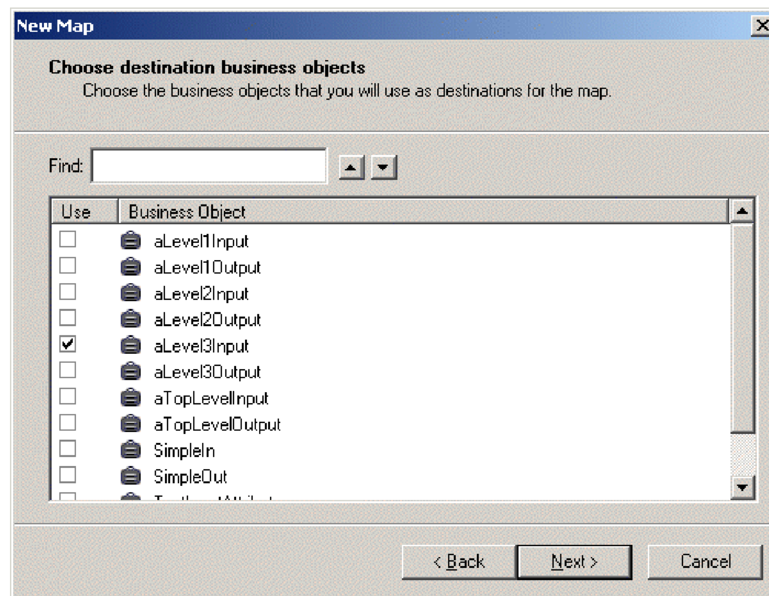


Figure 12. Selecting destination business objects

Tip: To locate a particular business object, enter its name in the Find field. The up and down arrows scroll through the business object list. Click Next to continue.

The New Map wizard does not require that you specify the destination business object. You can click Next without selecting the destination business object to postpone specifying this business object definition. You can specify it at a later time in the map workspace of the Diagram tab. For more information, see "Creating the source and destination business objects" on page 32.

Note: If you add or delete a business object from System Manager, it dynamically updates the list of business object definitions.

5. Specify the name to associate with the map.

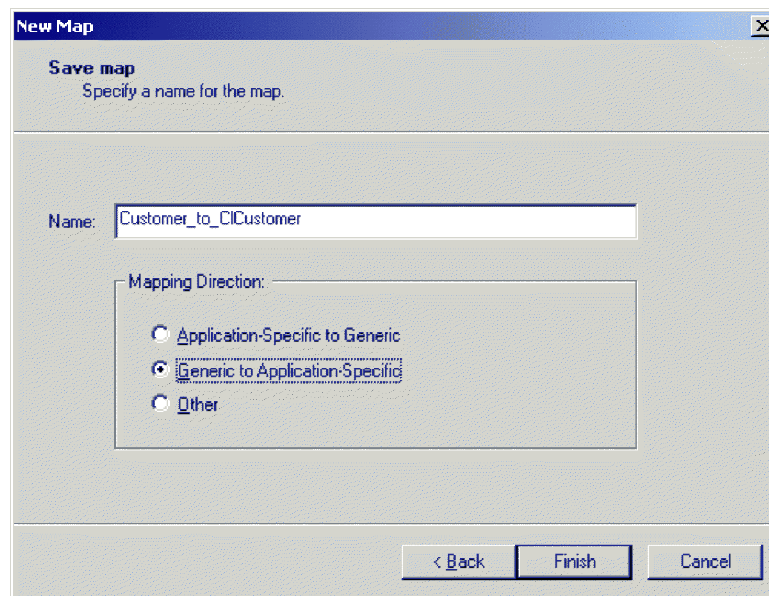


Figure 13. Saving new map

Rule: Map names can be up to 80 alphanumeric characters and underscores (_). Map Designer Express *does* enforce some naming restrictions. For example, it does *not* allow certain punctuation symbols, such as a period, a left brace ([), a right brace (]), a single quotation mark, a double quotation mark, or a space in the map name.

The New Map wizard does *not* require that you specify the map name. You can click Finish without entering the map name to postpone naming this map definition. When you save the map, Map Designer Express prompts you with the Save Map As dialog for you to specify the required map name. For more information, see "Saving a map to a project" on page 47.

Specify whether the map is an inbound or outbound map. This map role is needed for automatically generating relationship codes.

6. Click Finish to save the new map definition with the specified source and destination business objects.

   Result: Map Designer Express displays the new map's information in its Diagram tab.

Creating the source and destination business objects

If you do not specify the map's source and destination business objects from the New Map wizard, you can specify them from either of the following:

- From the Add Business Object dialog
- From the Diagram tab in the business object browser

From the Add Business Object dialog

You can add a source or destination business object to a map from the General tab of the Add Business Object dialog. You display the Add Business Object dialog in any of the following ways:

- Select Add Business Object from the Edit menu of Map Designer Express.

- In the Designer toolbar, click the Add Business Object button.
- From the Table tab, right-click in the empty area of the business objects pane and select Add Business Object from the Context menu.
- From the Diagram tab, right-click in the map workspace and select Add Business Object from the Context menu.

Through the General tab of the Add Business Object dialog, you specify the source and destination business objects. The General tab provides the following functionality:

- To specify a source business object:
  - Click the business object in the business object list.
  - Click the Add to Source button.
- To specify a destination business object:
  - Click the business object in the business object list.
  - Click the Add to Destination button.
- To locate a particular business object, enter its name in the Find field. The up and down arrows scroll through the business object list.
- To close the dialog, click Done.

From the map workspace

From the Diagram tab, you can add a source or destination business object to a map by dragging a business object definition from the business object browser onto the map workspace as follows:

- Drag the source business object to the left side of the map workspace. The business object displays and its title starts with `Src`.
- Drag the destination business object to the right side of the map workspace. The business object displays and its title starts with `Dest`.

Note: A dotted-line boundary divides the left and right halves of the workspace and identifies the source and destination portions of the map workspace. Be sure to carefully drop objects in the appropriate place.

Figure 14 shows the source and destination business objects in the map workspace.

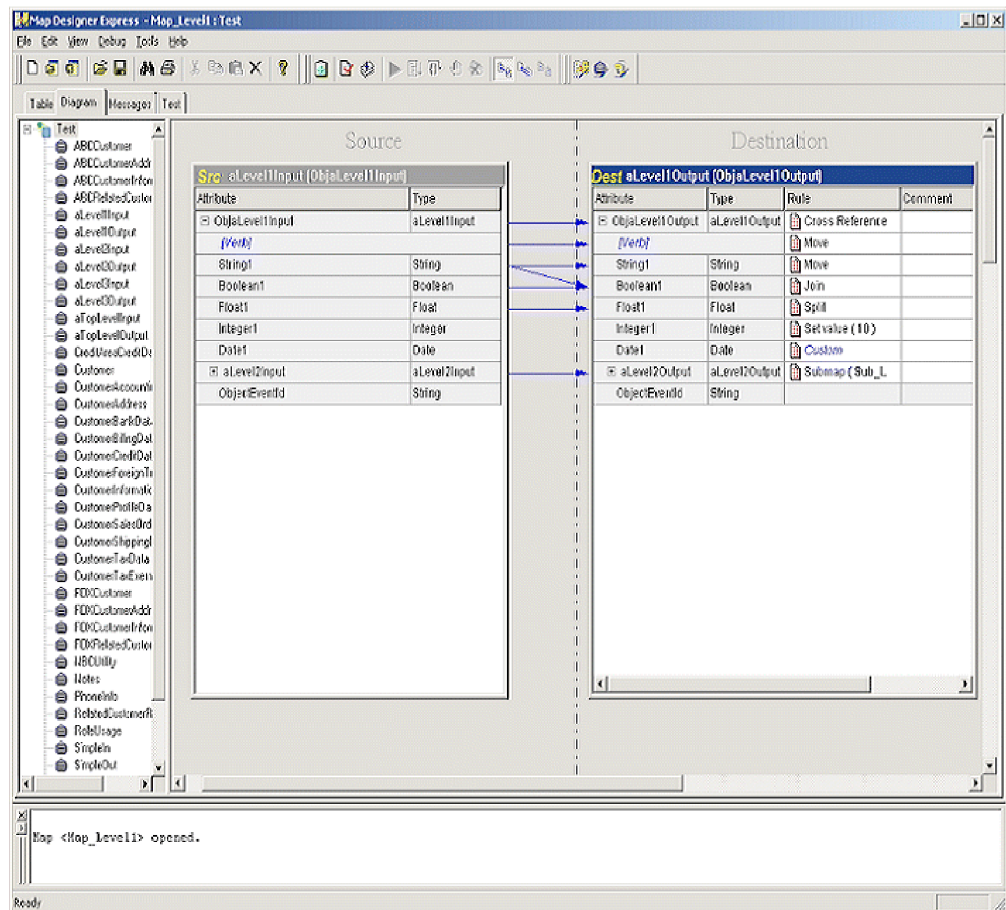


Figure 14. Defining Source and Destination business objects

Tip: Alternatively, you can create the source and destination business objects by right-clicking the business object in the business object browser; selecting Copy from the Context menu; then right-clicking in the map workspace and selecting Paste As Input Object or Paste As Output Object.

Map Designer Express creates a window, called a *business object window*, for the source and destination objects. The title bar of this window displays the business object instance name. For help interpreting the title bar of the business object window, see "Using generated business object variables and attributes" on page 140.. The business object window for the source business object contains columns for the name and data type of each source attribute. The business object window for the destination business object contains columns for the name, data type, transformation rule (which identifies the transformation step), and an optional comment.

If you make a mistake by dragging the wrong business object or making it an output object instead of input, you can delete the object from the map workspace and try again. To delete a business object from the map workspace, you can either:

- Select the business object to delete and use the Delete Current Selection option from the Edit menu (or press the `Del` key).

- Right-click the title bar of the business object's window and select Delete from the Context menu.

Setting the destination business object verb

The verb indicates how the system should process the business object's data. When a map executes, the system needs to know what verb to assign to each destination business object it creates.

If a map has only one source business object and one destination business object, the verb for the destination business object is usually the same as the verb for the source business object.

In this case, you need to copy the verb from the source business object to the destination business object (see Figure 14 on page 34), by defining a *Move* transformation rule with the source attribute as the source business object's verb and the destination attribute as the destination business object's verb. For more information, see "Copying a source attribute to a destination attribute" on page 37.

Tip: You can also drag-and-drop the verb from the source business object to the destination business object to define the value of the verb.

If a map has a destination business object with a verb that is not found in the source business object, you need to set the verb to a constant value, by defining a *Set Value* transformation rule with the destination attribute as the destination business object's verb. In the Set Value dialog box, enter the constant verb value. For more information, see "Specifying a value for an attribute" on page 36.

Maps sometimes have more than one source or destination business object, and these objects can have several child business objects. In these cases, you must consider carefully which verb to assign to each destination business object. Some destination business objects might require some custom logic to set the verb based on the verbs of one or more source business objects.

Specifying standard attribute transformations

You can specify several standard attribute transformations interactively in Map Designer Express. Table 12 shows the standard transformations that you can specify in Map Designer Express.

Table 12. Common attribute transformations

Name	Transformation step	Purpose
Set Value	"Specifying a value for an attribute" on page 36	For an attribute in the destination business object that is not found in the source business object but is required in the destination application
Move	"Copying a source attribute to a destination attribute" on page 37	For an attribute that is the same in both the source and destination business objects
Join	"Joining attributes" on page 38	For an attribute in the destination business object that is a combination of several attributes in the source business object

Table 12. Common attribute transformations (continued)

Name	Transformation step	Purpose
Split	“Splitting attributes” on page 40	For an attribute in the destination business object that is either: • Only one part of an attribute in the source business object • Made up of several fields, but with different delimiters from those in the source business object
Submap	“Transforming with a submap” on page 41	For attributes in the source and destination business objects that contain child business objects
Cross-Reference	“Cross-referencing identity relationships” on page 45	For maintaining the identity relationships for the business objects
Custom	“Creating a Custom transformation” on page 46	For an attribute that requires transformations not provided by the automatically generated transformations

In the Diagram tab, you can select which attributes display in the business object windows with the options of the View-->Diagram menu. You can choose to display all attributes, only linked (mapped) attributes, or only unlinked (unmapped) attributes.

Tip: Attributes appear in the same order that they appear in the business object definition. To locate a particular attribute in a long list of attributes, select Find from the Edit menu (or use the keyboard shortcut of `Ctrl+F`). For more information, see “Finding information in a map” on page 60.

Specifying a value for an attribute

Some destination attribute values do not depend on a source attribute and can be filled in with a constant value. This is especially true if the destination business object contains many attributes that are not found in the source business object but are required in the destination application. Some examples of default values for attributes are `CustomerStatus = "active"` or `AddressType = "business"`.

This type of transformation is called a *Set Value* transformation. You set the value of a destination attribute with the Set Value dialog, shown in Figure 15. You can display the Set Value dialog from either of the following Map tabs:

- From the Table tab:
  - Select the destination attribute whose value you want to set.
  - Click Set Value from the list in the Transformation Rule column.
- From the Diagram tab:
  - Select the destination attribute whose value you want to set.
  - Click Set Value from the list in the Rule column of the destination business object.
- If a Set Value transformation is already defined, you can display the Set Value dialog to reconfigure the transformation, including modifying its transformation code in either of the following ways:
  - Double-click the corresponding cell of the transformation rule column.
  - Click the Set Value bitmap icon contained in the transformation rule column.

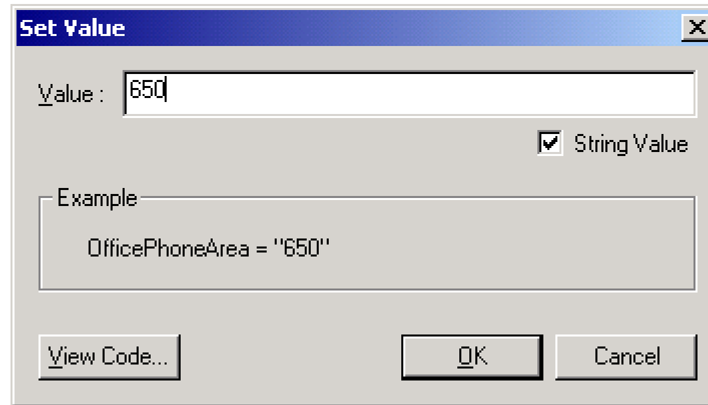


Figure 15. Set Value dialog

Through the Set Value dialog, you set the constant value to assign to the destination attribute. The Set Value dialog provides the following functionality:

- To specify the constant value, enter it in the Value field. For numeric values, simply enter the number and make sure that the String Value check box is not selected. For string values, enter the string value in the Value field and select the String Value check box.

  Note: The Set Value dialog uses the Examples area to show how the resulting destination attribute will look.
- To view the generated code, click View Code...

  Result: Map Designer Express brings up the Activity Editor in Java view, containing a sample of the transformation code in read-only mode for the destination attribute.
- To confirm the transformation setting, click OK.

Copying a source attribute to a destination attribute

The simplest kind of transformation step is a copy of one source attribute into a corresponding destination attribute. This type of transformation is called a *Move* transformation. You perform a move transformation from either of the following map tabs:

- From the Table tab:
  - Select the source attribute.
  - Select the destination attribute.
  - Click Move from the list in the Transformation Rule column.
- From the Diagram tab:
  - Select the source attribute.
  - Use `Ctrl+Drag` to move to the destination attribute; that is, hold down the `Ctrl` key and drag the attribute onto the destination attribute in the destination business object window. Continue to hold down the `Ctrl` key until after you release the mouse button; otherwise, the operation does not succeed.

    Map Designer Express creates a blue arrow from the source to the destination object. If the transformation involves a single source attribute that is *not* a

child business object, Map Designer Express assumes that the transformation is a Move and automatically assigns Move to the Rule column of the destination attribute.

Note: You can customize the key sequence used to initiate a Move transformation in the Diagram tab from the Key Mapping tab of the Preferences dialog. For more information, see "Specifying Key Mappings" on page 21.

Result: Map Designer Express generates the code to copy the value of the source attribute to the destination attribute. If the source and destination attributes are of different data types, Map Designer Express determines whether a type conversion is possible, and if so, generates the code to convert the source type to the destination type. If a type conversion is *not* possible, or might result in data loss, Map Designer Express displays a dialog box for you to confirm or cancel the operation.

If you want to see a sample of the generated code for the Move transformation, in the Context menu of the rule column, select View Source.

Joining attributes

You can concatenate, or join, the values from more than one source attribute into a single destination attribute. This type of transformation is called a *Join* transformation. For instance, the source business object might store the area code, telephone number, and extension in separate attributes, while the destination business object stores these values together in one attribute.

In addition to joining the attributes, you can reorder them and insert delimiters, parentheses, or other characters. For instance, when joining separate area code and telephone number attributes into a single attribute, you might want to insert parentheses around the area code.

Tip: The attributes you want to join can sometimes be located in more than one source business object, such as in a parent business object and one of its child business objects. You can also join an attribute with a variable you have defined. (To learn about defining variables, see "Using temporary variables" on page 142.)

You join multiple source attributes into one destination attribute with the Join dialog, shown in Figure 16. You display the Join dialog in either of the following ways:

- From the Table tab:
  - Select the source attributes to join.

    Tip: You can click Multiple Attributes in the combo box to display the Multiple Attributes dialog. In this dialog, you can check multiple source attributes. To locate a particular business object, enter its name in the Find field. The up and down arrows scroll through the business object list. Once you have selected the source attributes, click OK to close the dialog.
  - Select the single destination attribute.
  - Click Join from the list in the Transformation Rule column.
- From the Diagram tab:
  - Select two or more source attributes.

– Use Ctrl+Drag to move to the destination attribute; that is, hold down the Ctrl key and drag the selected source attributes to the destination attribute. Continue to hold down the Ctrl key until after you release the mouse button; otherwise, the operation does not succeed.

  If the transformation involves more than one source attribute, Map Designer Express assumes that the transformation is a Join. It automatically assigns Join to the Rule column of the destination attribute and displays the Join dialog.

Note: You can customize the key sequence used to initiate a Join transformation in the Diagram tab from the Key Mapping tab of the Preferences dialog. For more information, see “Specifying Key Mappings” on page 21.

If a Join transformation is already defined, you can use the Join dialog to reconfigure the transformation, including modifying its transformation code, in either of the following ways:

- Double-click the corresponding cell of the transformation rule column.
- Click the Join bitmap icon contained in the transformation rule column.

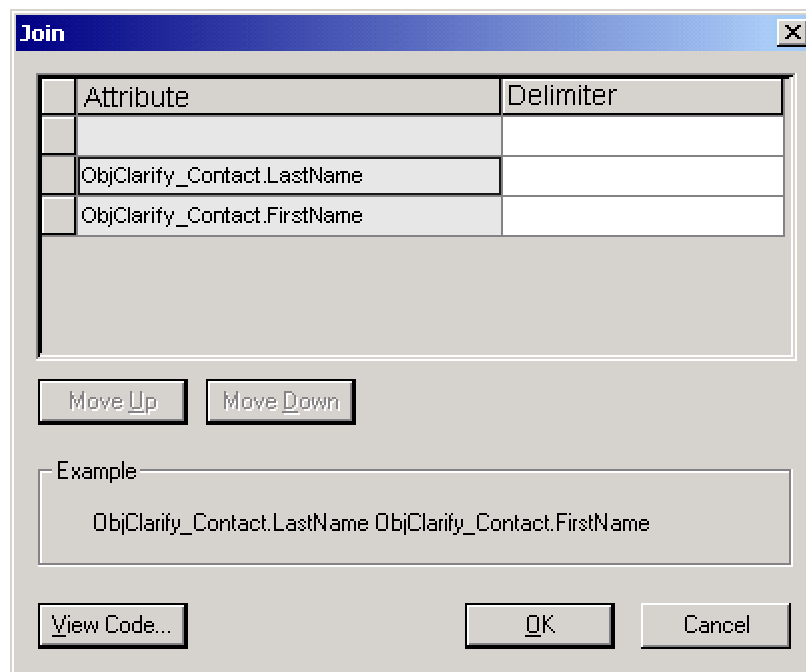


Figure 16. Join dialog

Through the Join dialog, you build an expression to concatenate the source attributes by adding delimiters, grouping with parentheses, and reordering the attributes if necessary. The Join dialog provides the following functionality:

- To insert a delimiter or parenthesis, enter it in the Delimiter field associated with the attribute. Do *not* put quotation marks around delimiters. The delimiter you enter is appended to the associated attribute. For leading delimiters, enter the delimiters in the Delimiters field of the initial blank line.

  Note: The Join dialog uses the Examples area to show how the resulting string will look after the join.
- To modify a delimiter or parenthesis you have entered, click in the Delimiter field and edit as appropriate.

- To reorder a delimiter or the attributes, click the left-most column to select the row, then click Move Up or Move Down to move the whole row up or down.
- To view the generated code, click View Code...

  Result: Map Designer Express brings up the Activity Editor in Java view, containing a sample of the transformation code in read-only mode for the destination attribute.
- To confirm the transformation setting, click OK.

Splitting attributes

To split a source attribute into two or more destination attributes, you specify the transformation for each destination attribute separately. This type of transformation is called a *Split* transformation. For instance, to split a source attribute, such as `phone_number`, into three separate destination attributes, such as `area_code`, `tel_number`, and `extension`, you specify the transformations for `area_code`, `tel_number`, and `extension` separately.

You split a source attribute into multiple destination attributes with the Split dialog, shown in Figure 17. You display the Split dialog in any of the following ways:

- From the Table tab:
  - Select the single source attribute to split.
  - Select one of the desired destination attributes.
  - Click Split from the list in the Transformation Rule column.
  - Repeat these steps for each destination attribute that receives a segment of the source attribute.
- From the Diagram tab
  - Select the single source attribute to split.
  - Use `Alt+Drag` to move to one of the destination attributes; that is, hold down the `Alt` key and drag the source attribute to one of the destination attributes.

    If the transformation involves more than one destination attribute, Map Designer Express assumes that the transformation is a Split. It automatically assigns Split to the Rule column of the destination attribute and displays the Split dialog.
  - Repeat these steps for each destination attribute that receives a segment of the source attribute.

  Note: You can customize the key sequence used to initiate a Split transformation from the Key Mapping tab of the Preferences dialog. For more information, see "Specifying Key Mappings" on page 21.

If a Split transformation is already defined, you can use the Split dialog to reconfigure the transformation, including modifying its transformation code, in either of the following ways:

- Double-click the corresponding cell of the transformation rule column.

- Click the Split bitmap icon contained in the transformation rule column.

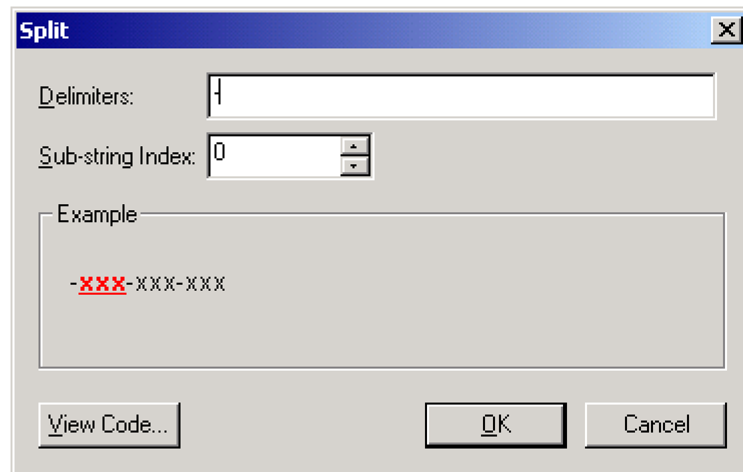


Figure 17. Split dialog

Through the Split dialog, you split an expression into segments that are separated by a delimiter. Each segment is identified with an index number, with the first segment having an index number of zero (0). The Split dialog provides the following functionality:

- To identify the delimiter by which to parse the source attribute, enter it in the Delimiter field. Do *not* put quotation marks around delimiters. You can specify one or more delimiters in this field. The transformation uses each of the specified delimiters to parse the string into segments. For example, to split `LastName,FirstName`, specify "**,**" as the delimiter, `LastName` as segment 0 (the first segment) and `FirstName` as segment 1 (the second segment).

  Note: The Split dialog uses the Examples area to show how the source attribute string looks and to indicate which segment is currently being accessed. The accessed segment displays in bold and red.
- To modify a delimiter or parenthesis you have entered, click in the Delimiter field and edit as appropriate.
- To identify the segment of the source attribute that is copied to the destination attribute, enter its index number in the Sub-string Index field.
- To view the generated code, click View Code...

  Result: Map Designer Express brings up the Activity Editor in Java view, containing a sample of the transformation code in read-only mode for the destination attribute.
- To confirm the transformation setting, click OK.

Transforming with a submap

A *submap* is a map that is called from within another map, called the *main map*. This section provides the following information about submaps:

- "Uses for submaps"
- "Specifying a Submap transformation" on page 43

Uses for submaps

You can call a submap to obtain a value for any destination attribute, but submaps are most commonly used for the following:

- To modularize a map
- To specify transformations between child business objects

Improving map modularity: Using submaps can improve the modularity of your maps by isolating common transformations that can be reused in more than one map. For example, a `Customer` business object might have an `Address` child business object that is also a child of an `Order` business object. If you create a submap for the `Address` business object, you can reuse the submap in both the `Customer` and `Order` business object maps.

Figure 18 illustrates how a submap, `MyAddrToGenAddr`, can be reused by two different maps.

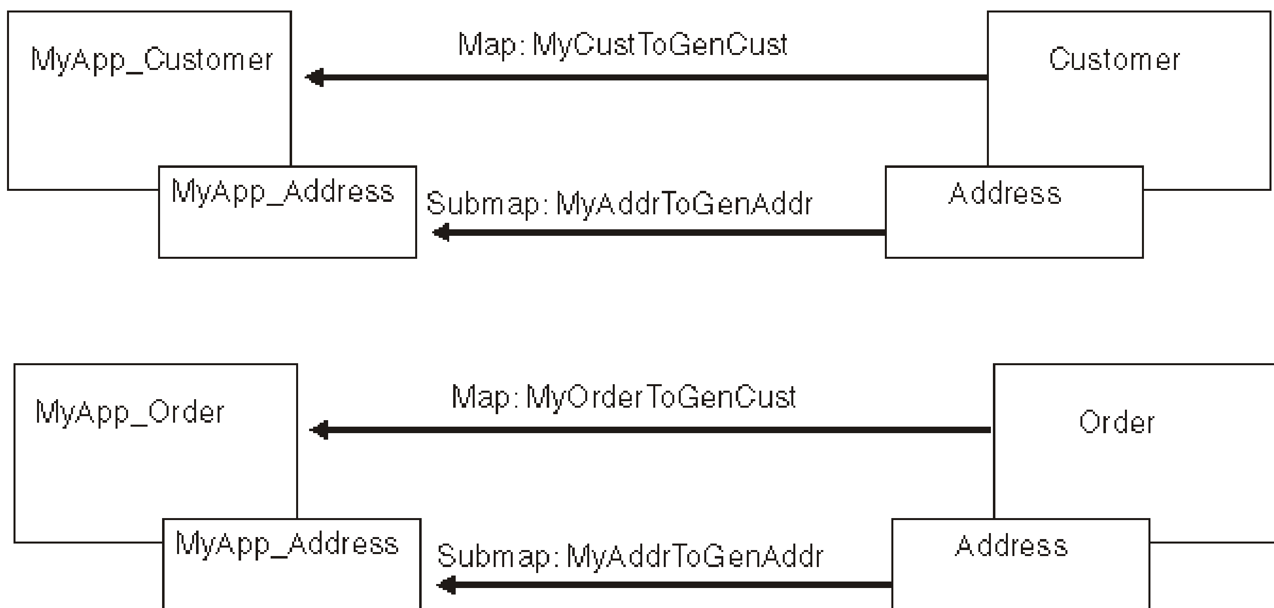


Figure 18. Using submaps for modularity

Transforming child business objects: When the source and destination attributes contain multiple-cardinality child business objects, it is useful to use a submap to specify their transformations. Typical examples of multiple-cardinality child business objects are the multiple addresses of a customer or the multiple line items in an order.

In the simplest case, you transform each source child business object into a single destination child business object, in a one-to-one relationship. Figure 19 illustrates the use of submaps for an `Employee` business object and its child business array that contains instances of `EmployeeAddress`.

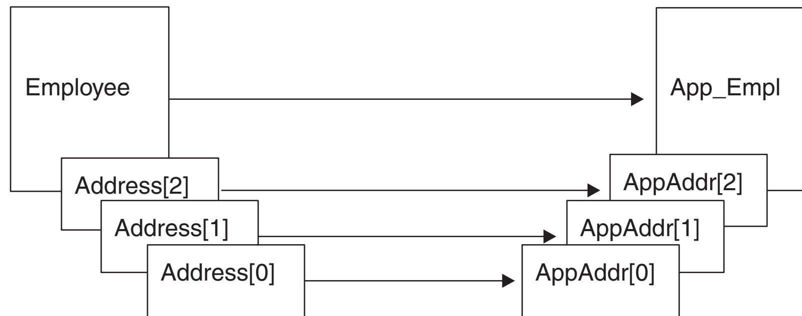


Figure 19. One-to-one transformation of child business object arrays

A submap can be associated with a conditional statement that governs whether it executes. For example, consider Figure 20: the `Order` business object has an `OrderLine` attribute that contains a multiple-cardinality child business object, `OrderLine`. The `OrderLine` business object has a `DeliverySchedule` attribute that contains a multiple-cardinality child business object, `DelSched`.

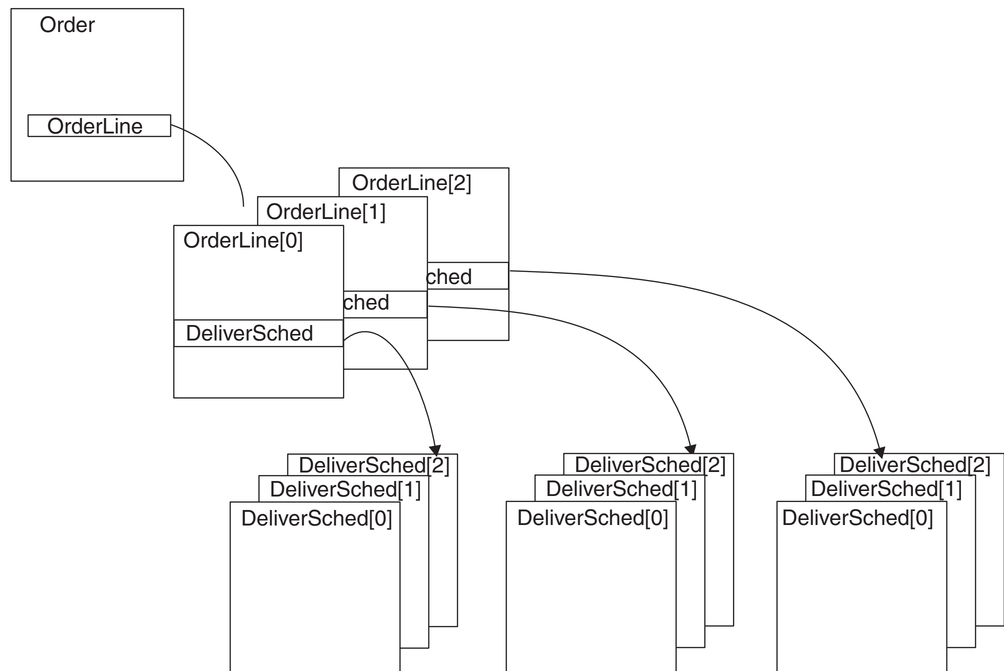


Figure 20. Source business object with multiple-cardinality child business object

Some conditions that can be written in the map for `Order` can:

- Execute the submap that transforms the `OrderLine` attribute in `Order` only if a different attribute in `Order` has a particular value.
- Execute the submap that transforms the `DeliverSched` attribute in `OrderLine` only if a different attribute in `OrderLine` has a particular value.
- Execute the submap that transforms the `DeliverSched` attribute in `OrderLine` only if an attribute in `Order` has a particular value.

Specifying a Submap transformation

Perform the following steps to create a Submap transformation.

1. Create the map that you want to use as a submap.

You do this in the same way that you create and save any other map. IBM naming conventions suggest that submap names begin with the string "`Sub_`".

2. Save the submap to the project in System Manager and compile the submap.
3. Specify the Submap transformation on the attribute in the parent business object that needs to call the submap. This source attribute contains a child business object that is mapped to a destination attribute that contains a child business object.

You specify that a submap needs to be called with the Submap dialog, shown in Figure 21. You display the Submap dialog in either of the following ways:

- From the Table tab:
  - In the parent map, select a source attribute (which is a child business object).
  - Select the desired destination attribute (which is also a child business object).
  - Click Submap from the list in the Transformation Rule column.
  - Repeat these steps for each source attribute that is a source business object for the submap and each destination attribute that is a destination business object for this submap.
- From the Diagram tab
  - In the parent map, select the source attribute (which is a child business object).
  - Use `Ctrl+Drag` to move to the destination attribute; that is, hold down the `Ctrl` key and drag the source attribute onto the destination attribute. Continue to hold down the `Ctrl` key until after you release the mouse button; otherwise, the operation does not succeed.

    If the transformation involves a source attribute that is a child business object, Map Designer Express assumes that the transformation is a Submap. It automatically assigns Submap to the Rule column of the destination attribute and displays the Submap dialog.

Note: You can customize the key sequence used to initiate a Submap transformation from the Key Mapping tab of the Preferences dialog. For more information, see "Specifying Key Mappings" on page 21.

If a Submap transformation is already defined, you can use the Submap dialog to reconfigure the transformation, including modifying its transformation code, in either of the following ways:

- Double-click the corresponding cell of the transformation rule column.

- Click the Submap bitmap icon contained in the transformation rule column.

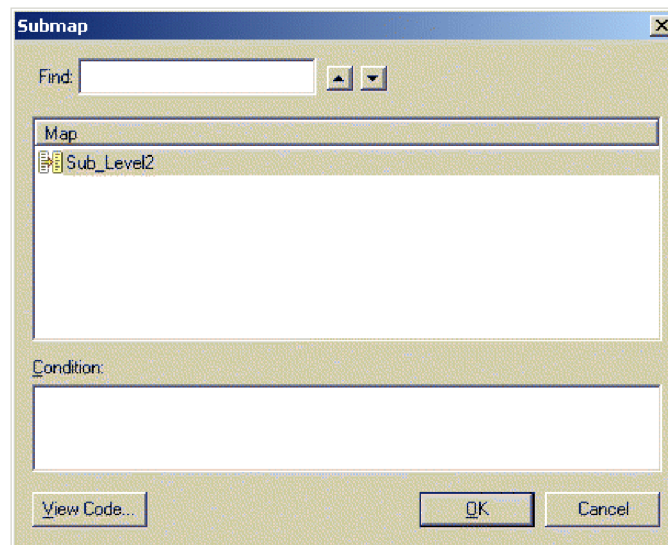


Figure 21. Submap dialog

Through the Submap dialog, you specify the name of the submap to call. The Submap dialog provides the following functionality:

- To identify the submap to call, select its name from the list in the Map area. The map list displays maps that meet the following conditions:
  - The submap has the same business object definitions for its source and destination business objects as the source and destination attribute you have selected.

  Tip: To locate a particular submap, enter its name in the Find field. The up and down arrows scroll through the business object list.
- To specify a condition for the submap, enter it in the Condition area of the Submap dialog. You can enter the condition now or simply dismiss the dialog and enter the condition in the destination attribute's generated code.
- To view the generated code, click View Code...

  Result: Map Designer Express brings up the Activity Editor in Java view, containing a sample of the transformation code in read-only mode for the destination attribute.
- To confirm the transformation setting, click OK.

Cross-referencing identity relationships

In some cases, the source attribute may need to reference a relationship table to find out what value to set in the destination attribute. This can be done using a *Cross-Reference* transformation.

Perform the following steps to use a cross-reference transformation:

1. Select the source and destination attributes in any of the ways previously described for other transformation. Both have to be business objects.
2. Select Cross-Reference in the corresponding transformation cell.

Result: The Cross-Reference dialog appears:

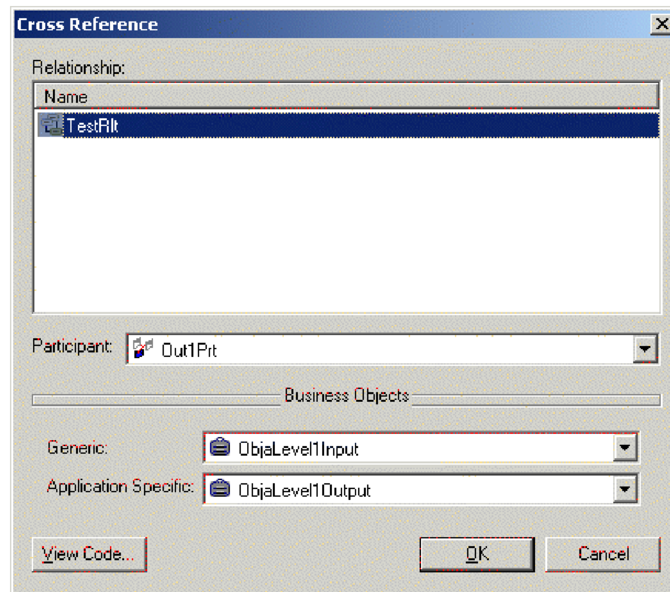


Figure 22. Cross-Reference dialog

3. In this dialog, select the relationship name from the list.

   Result: The Participant combo box will be populated with all participants from the selected relationship. The Business Object combo box, by default, will be populated according to the mapping role defined in the map property. You can change the combo boxes.

Creating a Custom transformation

In a *Custom* transformation, you use the Activity Editor to customize the activity for the transformation graphically. Perform the following steps to define a custom transformation from either the Table or Diagram tab:

- From the Table tab
  – Select the source attribute.
  – Select the desired destination attribute.
  – Click Custom from the list in the Transformation Rule column.
- From the Diagram tab
  – Select the source attribute.
  – Select the desired destination attribute.
  – Drag the source attribute onto the destination attribute in the destination business object window.

Note: You can customize the key sequence used to initiate a Custom transformation from the Key Mapping tab of the Preferences dialog. For more information, see "Specifying Key Mappings" on page 21.

Result: Map Designer Express displays the Activity Editor with a graphical view. For more information on the Activity Editor, see "Using the Activity Editor" on page 87.

If a custom transformation is already defined, you can modify its transformation code in either of the following ways:

- Double-click the corresponding cell of the transformation rule column.
- Click the Custom bitmap icon contained in the transformation rule column.

Saving maps

To preserve the map definition for use at a later time, you must save the map. Before Map Designer Express saves a map, it first validates the map. For more information, see "Validating a map" on page 69.

Map Designer Express provides two ways to save the current map:

- "Saving a map to a project" on page 47
- "Saving a map to a file" on page 48

Important: For Map Designer Express to be able to save a map, a map must currently be open.

Saving a map to a project

A map definition stores map information in a project in System Manager. This map definition contains the following information for a map:

- The general map information, which includes map properties
- The map design, which includes the transformation mappings.
- The custom transformation code

To save a map to a project in System Manager, you can perform any of the actions shown in Table 13..

Table 13. Saving a map to the project

If you want to . . .	Then . . .
Save the map definition to the name of the currently open map.	Use any of the following: • Select To Project from the File --> Save submenu. • Use the keyboard shortcut `Ctrl+S`. • In the Standard toolbar, click the Save Map to Project button).
Save the map definition to a name different from the currently open map.	Use any of the following: • Select To Project from the File--> Save As submenu. • Use the keyboard shortcut `Ctrl+Alt+S`. **Result:** Map Designer Express displays the Save Map As dialog in which you can specify the map name.

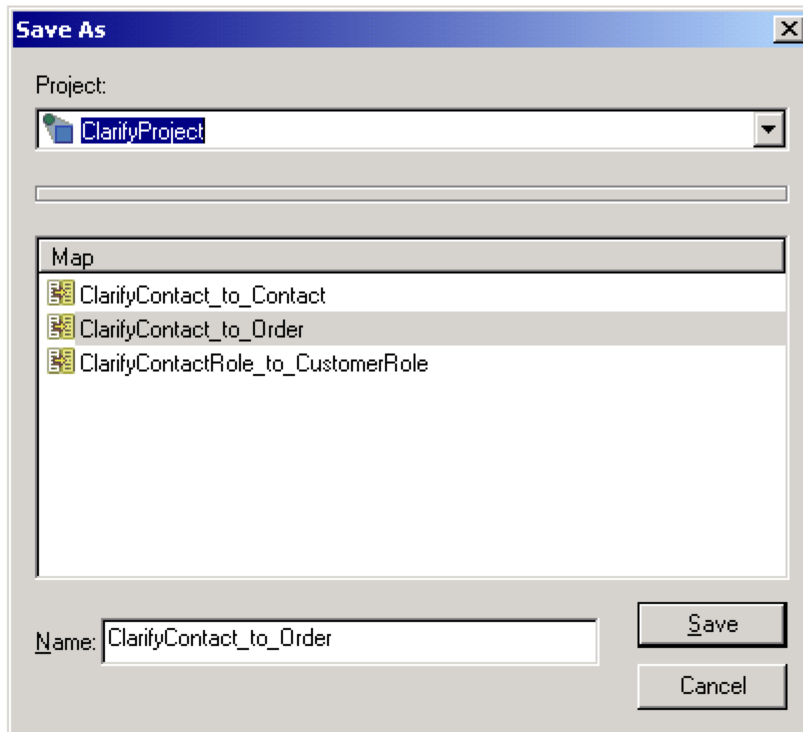


Figure 23. Save As dialog

When you save the map, Map Designer Express saves the map definition and map content to the project in System Manager. It saves the map content as XML data.

Note: You can specify whether Map Designer Express automatically saves a map to the project in System Manager before compiling the map with the option `Compile Map: save map before compile`. By default, this option is enabled. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.

Tip: To rename an existing map, select To Project from the File--> Save As submenu.

Saving a map to a file

A map definition can be stored as text in an operating-system file, called a *map definition file*. A map definition file contains the complete map definition; that is, this file uses Extended Markup Language (XML) format to represent the following parts of a map definition:

- The general map information, which includes map properties
- The map content, which includes the transformation mappings in an uncompressed format

Recommendation: Map Designer Express creates a map definition file with a `.cwm` extension. You should follow a naming convention for your map definition files, such as. using the file extension (`.cwm`) to distinguish them.

You import a map definition into Map Designer Express by opening an existing map definition file. For more information, see "Opening a map from a file" on page 53.

You can save the currently open map to a map definition file in any of the ways shown in Table 14.

Table 14. Saving a map to a map definition File

If you want to . . .	Then . . .
Save the map to the name of the currently open map in the format: `MapName.cwm` (where *MapName* is the name of the currently open map) **Note:** Map Designer Express will always open the File Save dialog if you do not open the currently opened map from file.	Use any of the following: • Select To File from the File--> Save submenu. • Use the keyboard shortcut `Ctrl+E`. • In the Standard toolbar, click the Save Map to File button (see Figure 23).
Save the map to a specified map definition file. Map Designer Express displays a dialog box to allow you to select the file name.	Use either of the following: • Select To File from the File--> Save As submenu. • Use the keyboard shortcut `Ctrl+Alt+F`.

Note: When you select the To File option from the File-->Save or File-->Save As menus, Map Designer Express displays a dialog box to allow you to select the file name. This file name identifies the file. It is not necessarily the name of the map.

Example: You can save `MapA` in a file named `fileA.cwm`. This `fileA` file contains the map definition for `MapA`. When Map Designer Express opens the `fileA` map definition file, it displays the `MapA` map definition.

Tip: Exporting a map copies only the map.

Checking completion

When you are mapping two large business objects, it is easy to overlook some required attributes. You can search for attributes that are not yet mapped to make sure that you have specified all desired transformations. Such attributes are called *unlinked attributes*.

Perform the following step to check completion:

- Select Find from the Edit menu and click the Unlinked attributes option in the Find control pane.

  Result: Map Designer Express displays a list of attributes for which there is no transformation code. For more information, see "Finding information in a map" on page 60.

Note: Once the code is completed, you must compile and test it. For information on compiling a map, see "Compiling a map" on page 70. For information on testing a map, see "Testing maps" on page 72.

Mapping standards

This section provides the following procedural standards for maps:

- "Tips on mapping individual attributes"
- "Setting comments in the comment field of the attribute"

Tips on mapping individual attributes

If the attribute mapping does *not* include relationship management, copy the source attribute to the destination attribute (see "Copying a source attribute to a destination attribute" on page 37).

Important: Do *not* map the `ObjectEventid` attribute. InterChange Server Express reserves the `ObjectEventId` for its own processing purposes. Any custom code that has `ObjectEventId` as destination attribute will not execute properly.

Setting comments in the comment field of the attribute

Attribute comments can improve the readability of your map. However, Map Designer Express does *not* automatically generate a comment for an attribute. Table 15 provides some suggested standards for attribute comments based on the type of transformation associated with the destination attribute.

Table 15. Settings for the Attribute Comment

Situation	Setting for Attribute Comment
If the child business object is *not* mapped	`=No mapping`
Set Value transformation	`=SET VALUE(`*value*`)`
Move transformation	`=MOVE`
Join transformation	`=JOIN(`*srcAttr1*`, `*srcAttr2*`, ...)`
Split transformation	`=SPLIT(`*srcAttr*`[`*index*`])`
For child business objects, when the mapping is done *without* calling a submap to indicate the object has to be expanded to see its attributes	`=Mapping here`
If the code to call the submap is generated	`=SUBMAP(`*mapName*`)`
Custom transformation that is *not* one of those listed above	`=CUSTOM(`*summary*`)`
If the attribute's code does not contain anything except setting the verb	`=SET VERB`

Chapter 3. Working with maps

This chapter describes some advanced features of Map Designer Express that you might use after creating maps.

The chapter covers the following topics:

- "Opening and closing a map" on page 51
- "Providing map property information" on page 54
- "Using map documents" on page 56
- "Finding information in a map" on page 60
- "Finding and replacing text" on page 62
- "Printing a map" on page 62
- "Deleting objects" on page 63
- "Using execution order" on page 66

Opening and closing a map

Map Designer Express displays one map at a time within the tab window. This map is called the *current map* (sometimes called the "currently open map"). You can control which map is the current map with the following Map Designer Express procedures:

- "Opening a map"
- "Closing a map" on page 53

Opening a map

A map must be open in Map Designer Express before you can view its information in a Map tab or modify this information. When Map Designer Express opens a map, if the `validate map when open` preference is enabled, it first performs a set of validations on this map.

Note: You can specify whether Map Designer Express validates a map when it opens it, with the option `Open Map: validate map when open`. By default, this option is enabled.

If this preference is enabled when a map that uses big business objects (that is, thousands of attributes) is opened, Map Designer Express may take a long time to open the map. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.

The validations that Map Designer Express performs on the map are as follows:

- Ensures that each business object definition that the map uses is defined in the project in System Manager.
- Ensures that every attribute in the map exists in the specified business object definition, as defined in the project in System Manager.
- Ensures that the type of each attribute in the map matches its type in the specified business object definition, as defined in the project in System Manager.
- Validates transformations:

© Copyright IBM Corp. 2003

– Ensures execution order is correct; that is, that execution order is unique, positive, and consecutive.
– Ensures that no attributes have cyclic dependencies on each other. If any cyclic transformations are found, Map Designer Express displays the cyclic rules in the output window.
– Checks transformation information:

Move transformation—only one source attribute is involved.

Join transformation—more than one source attribute is involved.

Split transformation—only one source attribute is involved; split index is greater than or equal to zero; split delimiter is not empty.

Set Value transformation—no source attribute is involved; a value has been specified.

Submap transformation—at least one source attribute is involved; submap name is specified.

Cross-Reference transformation—only one source attribute is involved.

Map Designer Express provides the following ways to open a map:
- "Opening a map from a project in System Manager" on page 52
- "Opening a map from a file" on page 53

Opening a map from a project in System Manager

Perform the following steps to open a map from a project in System Manager:

1. Open the Open a Map from a Project dialog in any of the following ways:
   - Select From Project from the File-->Open submenu.
   - Use the keyboard shortcut of `Ctrl+O`.
   - In the Standard toolbar, click the Open Map from Project button.

   Result: Map Designer Express displays the Open Map dialog.

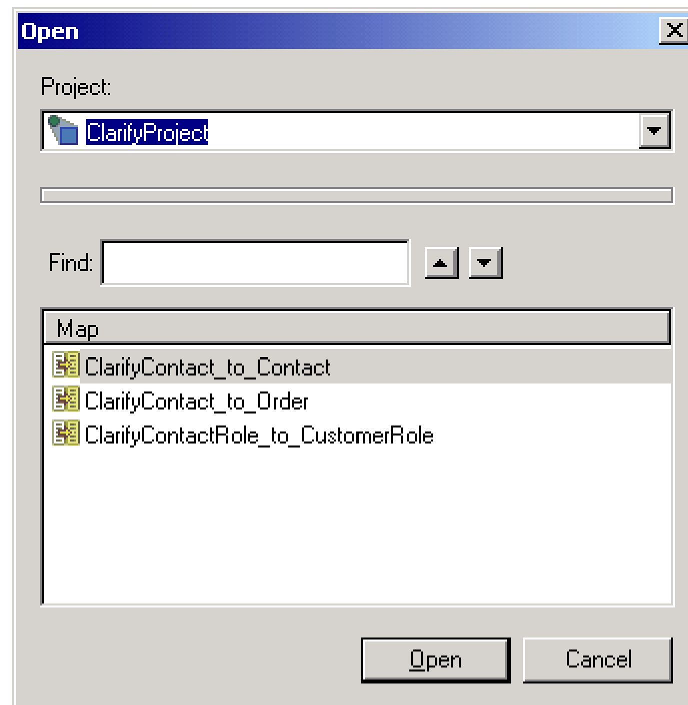


Figure 24. Open Map dialog

2. Select the project.
3. Select the map's name from the list of maps currently defined in the project in System Manager.

   Tip: To locate a particular map name, enter its name in the Find field. The up and down arrows scroll through the map list.
4. Click the Open button to open the map from the project.

Opening a map from a file

A map definition can be stored in XML format in an operating-system file called a *map definition file*. To create a map definition file, you save the map as a map design file (`.cwm`) in Map Designer Express. For more information, see "Saving a map to a file" on page 48.

When you open a map definition file, you open the map in Map Designer Express.

Perform the following steps to open a map definition file:

1. Open the Open a Map from a File dialog in any of the following ways:
   - Select From File from the File--> Open submenu.
   - Use the keyboard shortcut of `Ctrl+I`.
   - In the Standard toolbar, click the Open Map from File button.

   Result: The Open file with Map dialog box appears.
2. Select the map definition file you want to open. The file must be a `.cwm` file created by Map Designer Express.

   Result: Map Designer Express opens the map definition file. The map information appears in the Map tabs.

   Important: Opening the map in Map Designer Express does *not* automatically save the map to the project. To save this map to the project, continue to step 3.
3. Save the map to the project in System Manager. For more information, see "Saving a map to a project" on page 47.

   Rule: You must save the map to the project in System Manager for it to be compiled. To compile the map, select Compile from the File menu. For more information, see "Testing maps" on page 72.

Closing a map

Perform one of the following actions to close the current map, which is displaying in the tab window:

- Open a new map in any of the ways discussed in "Opening a map" on page 51.

  Result: Map Designer Express closes the current map before it opens a new one.
- Select Close from the File menu.

  Result: Map Designer Express closes the current map and clears the tab window. To make a new map current, you can either create a new map or open an existing map.
- Exit from Map Designer Express in any of the following ways:
  - Select Exit from the File menu.
  - Use the keyboard shortcut of `Alt+F4`.

Result: Map Designer Express automatically closes the current map before it exits.

Note: If you have changed the current map since it was last saved, Map Designer Express displays a confirmation box to confirm the map closure.

Providing map property information

Map Designer Express provides the Map Properties dialog (see Figure 25) to display and specify property information for a map. To display the Map Properties dialog, take any of the following actions:

- From the Edit menu, select Map Properties.
- Use the keyboard shortcut of `Ctrl+Enter`.
- In the map workspace of the Diagram tab, right-click and select Map Properties from the Context menu.

The Map Properties dialog provides the following tabs:

- General tab
- Business Objects tab

Figure 25 shows the General tab of the Map Properties dialog.

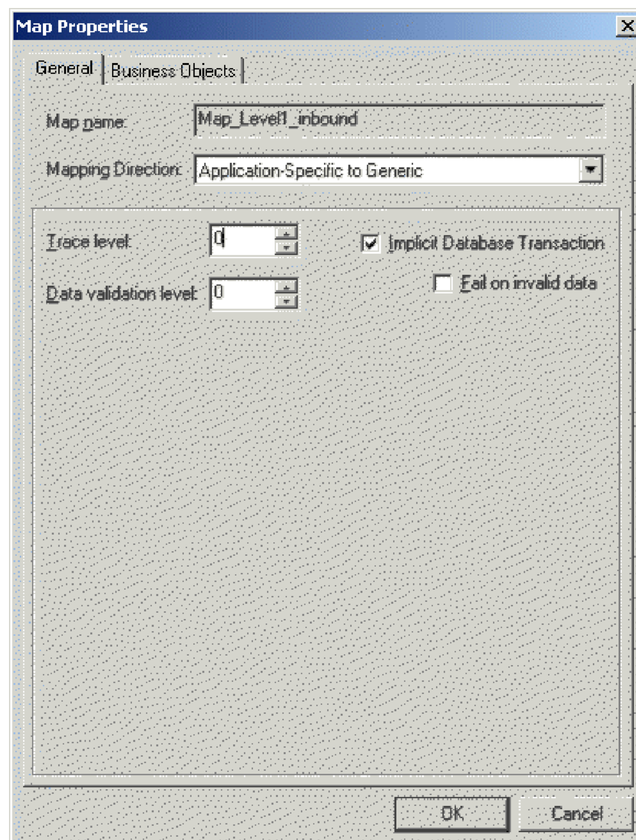


Figure 25. General tab of Map properties dialog

Defining General Property information

The General tab of the Map Properties dialog displays the general property information shown in Table 16.

Table 16. General Map Property Information

General Map Property	Description	For more information
Map name	Identifies the map whose properties the dialog displays. This field is initialized when you create a new map and is not an editable field.	N/A
Mapping role	Identifies the purpose of the map. Possible values of mapping roles are: • Application-specific to generic • Generic to application-specific • Other (for maps that do not have a specific mapping direction associated with them) **Note:** For previously defined maps that do not have this property information, the combo box will be empty. This is permissible as long as you do not use any Relationship transformation rules. When you first create a Relationship transformation rule and this value is empty, Map Designer Express will prompt you for this value.	
Run-time properties	These map properties (trace level, data validation level, implicit database transaction, and fail on invalid data) apply to the map instance at run time. You can specify these properties here in the General tab of Map Designer Express's Map Properties dialog or from the Map Properties window that System Manager provides. The changes are made to the local file system. Deploying the map to the server will not update the run-time instance. **Note:** You can update these map properties dynamically from the server component management view by right-clicking on a map and selecting the properties from the Context menu. The changes will be automatically updated to the server. For more information, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.	
Trace level	Sets the trace level for the map.	"Adding trace messages" on page 409
Data validation level	Allows you to check each operation in a map and log an error when data in the incoming business object cannot be transformed.	"Creating custom data validation levels" on page 146
Implicit Database transaction	Determines whether InterChange Server Express uses implicit transaction bracketing for transactions over its connections.	
Fail on invalid data	Determines whether map execution fails if data is invalid.	"Creating custom data validation levels" on page 146

Defining business objects

The Business Objects tab of the Map Properties dialog displays information about the map's business objects. It lists the source and destination business objects as well as any temporary business object that might be defined. For more information, see "Generating business object variables" on page 140.

Using map documents

Map Designer Express supports creation of a *map document*, which allows you to see all transformations in a single map or between two maps. While checking a map, you might want to view all of its transformations in a single operation, rather than opening and viewing each attribute separately. To do so, you can create a map document that contains all transformations. A map document provides you with an automated way to document native-map transformations.

This section provides the following information:

- A description of the two HTML files that make up a map document
- How to create a new map document
- How to view a map document
- How to print out a map document

What Is a map document?

A *map document* consists of two HTML files that describe all transformations of a map (or set of maps):

- A map-table file that describes the map transformations in a tabular format.

  The map-table file has the name `*mapDoc*.HTM`.
- A Java-code file that contains the code of the map transformations.

  The Java-code file has the name `*mapDoc*JavaCode.HTM`.

In both these HTML files, *mapDoc* is the user-specified name of the map document.

The map document can include information for all attributes, only those attributes that have map transformations, or only those attributes that do not have map transformations (unlinked attributes). If you specify all attributes, the map document also contains a list of unlinked attributes in the source and destination business objects.

The following sections describe the format of the two HTML files of a map document.

Map-table file format

The map-table file, *mapDoc*`.html`, describes the map transformations in a tabular format:

- If the map document describes *only one map*, Map Designer Express creates a single-map map table.
- If the map document describes *two maps*, Map Designer Express creates a multiple-map map table.

Single-map map table: A *single-map map table* describes the mapping flow in a single map; that is, it describes the transformations between a source and destination business object. The single-map map table has the following columns:

- `Source Attribute` shows the names of the source business object's attributes.
- `Transformation Rule` describes the kind of mapping transformation between the attribute in the source business object (in the column to the left) and the attribute in the destination business object (in the column to the right). The transformations listed in this column are hypertext links to the location of the attribute in the *mapDoc*`JavaCode.HTM` Java-code file for the map.

- `Destination Attribute` shows the names of the destination business object's attributes.

Figure 26 shows the HTML file that contains a single-map map table.

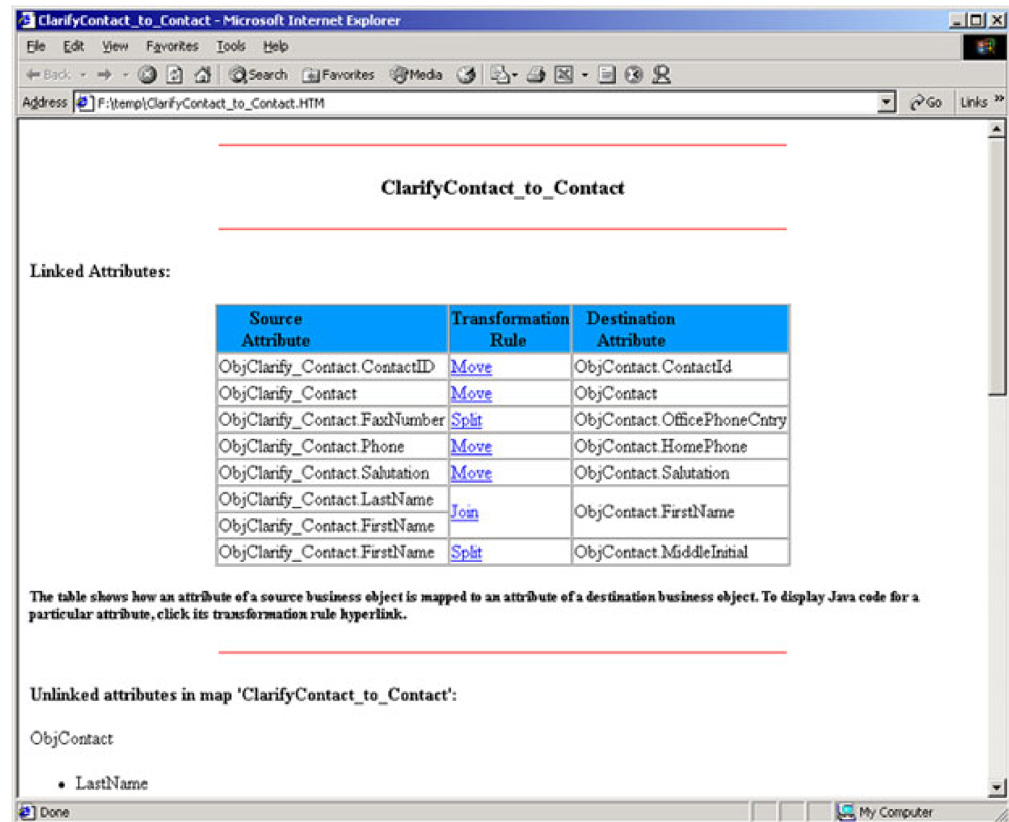


Figure 26. Single-map map table

Note: If you enabled the Comment check box Create Map Document dialog, the map table contains a fourth column called `Comment`, which shows the comment for each of the destination attributes in the table.

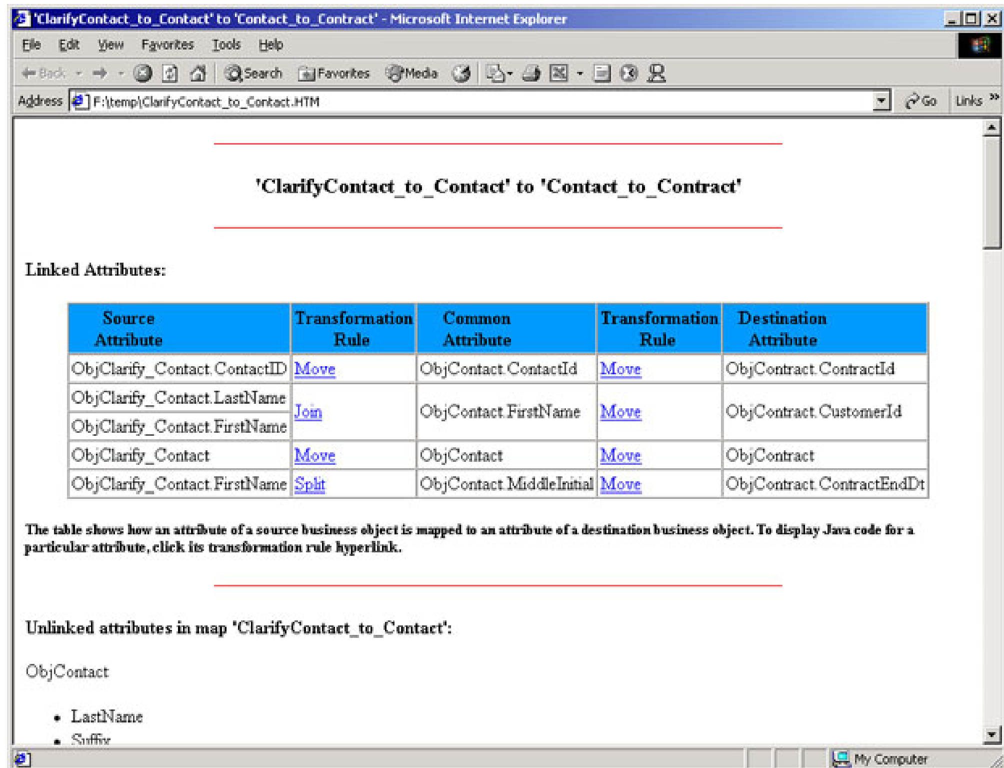
Multiple-map map table: A *multiple-map map table* describes the mapping flow between two maps; that is, it describes the transformations in the inbound map (between the application-specific and generic business object) and an outbound map (between the generic and application-specific business object). The multiple-map map table has the following columns:

- `Source Attribute` shows the names of the application-specific business object's attributes.
- The first `Transformation Rule` column describes the kind of mapping transformation between the attribute in the application-specific business object (in the column to the left) and the attribute in the generic business object (in the column to the right). The transformations listed in this column are hypertext links to the location of the attribute in the *mapDoc*`JavaCode.HTM` Java-code file for the inbound (application-specific to generic) map.
- `Common Attribute` shows the names of the generic business object's attributes.
- The second `Transformation Rule` column describes the kind of mapping transformation between the attribute in the generic business object (in the column to the left) and the attribute in the application-specific business object (in

the column to the right). The transformations listed in this column are hypertext links to the location of the attribute in the *mapDoc*`JavaCode.HTM` Java-code file for the outbound (generic to application-specific) map.

- `Destination Attribute` shows the names of the application-specific business object's attributes.

Figure 27 shows the HTML file that contains a multiple-map map table.



Source Attribute	Transformation Rule	Common Attribute	Transformation Rule	Destination Attribute
ObjClarify_Contact.ContactID	Move	ObjContact.ContactId	Move	ObjContract.ContractId
ObjClarify_Contact.LastName ObjClarify_Contact.FirstName	Join	ObjContact.FirstName	Move	ObjContract.CustomerId
ObjClarify_Contact	Move	ObjContact	Move	ObjContract
ObjClarify_Contact.FirstName	Split	ObjContact.MiddleInitial	Move	ObjContract.ContractEndDt

Figure 27. Multiple-map map table

Java-Code file format

The Java-code file, *mapDoc*`JavaCode.html`, provides more detailed information about the map. It contains the Java code that performs the transformations. This code is in standard program format. The Java-code file is useful when you want to view all map transformations in a single operation, rather than opening and viewing each attribute separately.

Creating a map document

Perform the following steps to create a map document:

1. Open the Create Map Document dialog by selecting Create Map Documents from the File menu.

   Result: Map Designer Express displays the Create Map Document dialog (see Figure 28).
2. Select the map-document configuration options from the Create Map Document dialog:
   - Specify the project.
   - Specify the maps that are involved in the map document.

Guideline: If you do *not* check the "Show mapping flow with two maps" check box, you can select only one map from the drop-down list. The drop-down list includes all maps currently defined. If a map is currently open, its name appears by default.

If you check the "Show mapping flow with two maps" check box, the second drop-down list is enabled. This second drop-down list provides only those maps that share the same generic business object as the first map. From this list, you can select the name of the second map to include in the map document.

- Specify the attributes in the destination business object to include in the map document.

  Click the appropriate radio button to indicate whether to include all attributes, only mapped attributes, or only unmapped attributes in the map document.
- Specify a name for the new map document.

  Guideline: You can click the Browse button to find a location for the map-document file. Map Designer Express automatically appends the suffix .HTM to the map-document name you enter. Therefore, you do not need to specify a file extension.

3. To initiate creation of the map document, select one of the following options:
   - Click Save to save the selected maps in a map document.
   - Click Save/View to save the selected maps in a map document and view this new map document in an HTML browser.

Figure 28 shows the Create Map Document dialog.

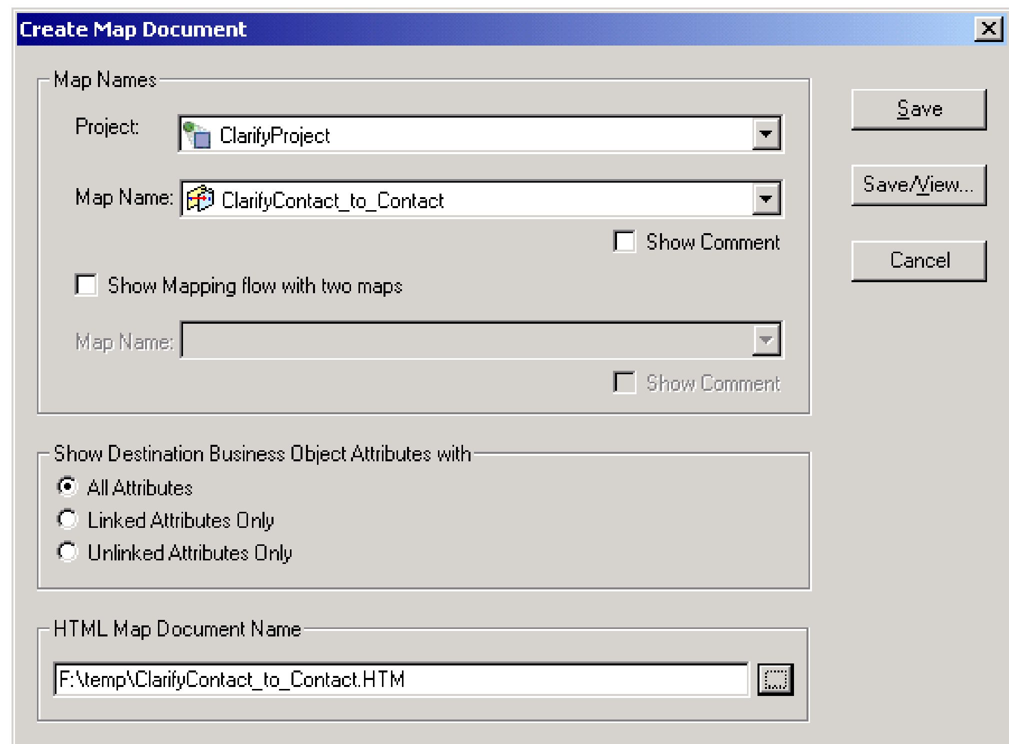


Figure 28. Create Map Document dialog

When you create a map document, Map Designer Express creates the map document as a Hypertext Markup Language (HTML) file (*mapDoc*.html) and a related Java code file (*mapDoc*JavaCode.html) where *mapDoc* is the map-document name you specified in the Map Document Configuration dialog.

Viewing a map document

You can view a map document in any of the following ways:

- Open an existing map document in either of the following ways:
  - Select the View Map Document option of Map Designer Express's File menu.
  - Use the keyboard shortcut `Ctrl+M`.

  Result: The Open dialog displays the available map-document files. Specify the HTML map document to read and click Open.
- Open a new map document by clicking Save/View on the Map Document Configuration dialog.
- Go into the directory that contains the map document files and double-click the desired file.

Result: Map Designer Express invokes your browser to display the HTML map-document file that you selected.

In addition, you can view the Java code associated with a particular transformation by clicking the entry in the Mapping Action column of the map table. Your browser displays the corresponding Java code segments that implement the mapping between the associated source and destination attributes.

Printing a map document

Perform the following steps to print a map-document file:

1. View the desired file in your HTML browser.

   For more information, see "Viewing a map document" on page 60.
2. Print the displaying HTML file from the browser.

   Select the Print option of the browser's File menu, use the keyboard shortcut (`Ctrl+P`), or select the Print icon from the Standard tool bar.

Finding information in a map

You can use Map Designer Express's search facility to perform the following searches:

- Search for text in an attribute name or in the attribute's transformation code.
- Search for unlinked attributes.

Perform the following steps to find information in a map.

Initiate a find in any of the following ways:

- Select Find from the Edit menu.
- Use the keyboard shortcut `Ctrl+F`.
- In the Standard toolbar, click the Find button.

Result: Map Designer Express displays the Find control pane (see Figure 29)

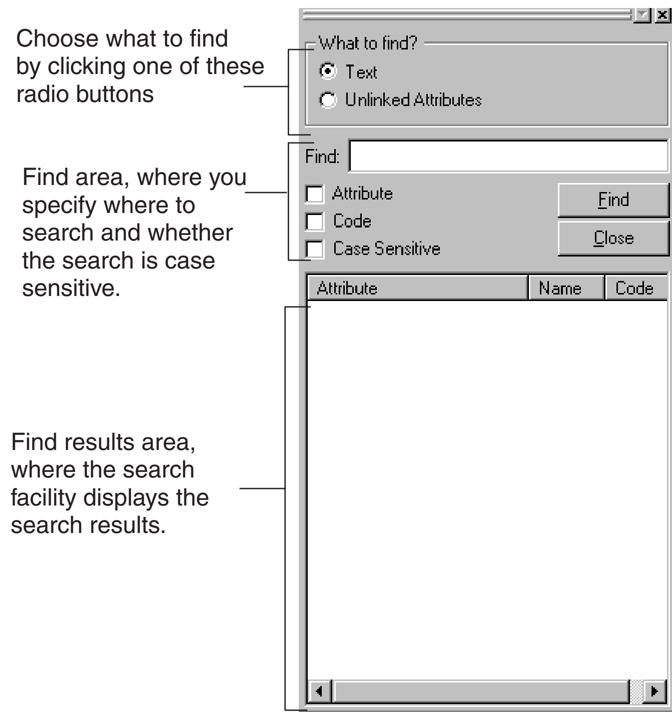


Figure 29. Find Control Pane

From the Find control pane, select one of the radio buttons in the What to find? area to indicate which kind of search you want to perform:

- To search for text:
  - Click the Text radio button.
  - Enter the text to search for in the Find field. You can enter multiple words and spaces if necessary.
  - Indicate where to search for the text by selecting one or more options in the Find area:

    Attribute—search the attribute names for the specified text.

    Code—search the attributes' transformation code for the specified text. You can select either Attribute or Code, or both of those options.

    Case Sensitive—make the text search case sensitive. To find only instances of the text that have the same case that you typed, select Case Sensitive.

    Restriction: You cannot search on data types or comments.
  - Click Find to initiate the search.
- To search for unlinked attributes:
  - Click the Unlinked Attributes radio button. The Find control pane deactivates the fields in the Find area.
  - Click Find to initiate the search.

Result: Map Designer Express displays the search results in the Find Results area. You can click any attribute name to automatically select that attribute in the map. Click Close to close the Find control pane.

Finding and replacing text

Using Map Designer Express's Find and Replace capability, you can search for specified text in the comments of a transformation rule and replace it with other specified text.

Perform the following steps to find and replace text.

1. Initiate a find and replace in any of the following ways:
   - Select Replace from the Edit menu.
   - Use the keyboard shortcut `Ctrl+H`.

   Result: Map Designer Express displays the Replace dialog.

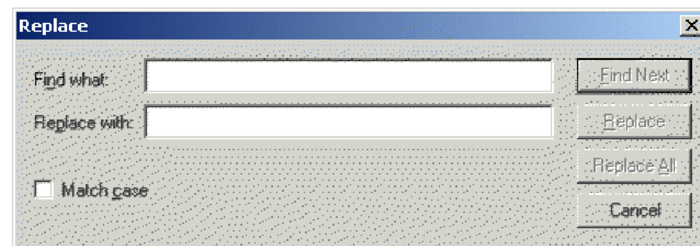


Figure 30. Replace dialog

2. In the Replace dialog, enter the text to search for in the Find what field and the text to replace it in the Replace with field. Select Match case, as necessary.
3. Click Find Next to initiate the search.

   Result: The Table view will be activated and the text will appear in the comment column in the Table view.
4. Click Replace to replace the match with the new text.

   Guideline: You can replace all similar matches with one action by clicking Replace All.
5. To continue finding and replacing the specified text, instance by instance, repeat steps 3 and 4.

Printing a map

Map Designer Express allows you to print a map. It creates a tabular representation of the map, much like the map appears in the Table tab. You can print a map in any of the following ways:

- Select Print from the File menu to print the current map.
- Use the keyboard shortcut of `Ctrl+P`.
- In the Standard toolbar, click the Print button.

Map Designer Express also supports the following standard print tasks:

- Print Preview—select Print Preview from the File menu to preview the page layout for the current map.
- Print Setup
  - Select Print Setup from the File menu to display the Print Setup dialog, where you can configure information such as printer setting, paper size and orientation.
  - Use the keyboard shortcut of `Ctrl+Shift+P`.

Guideline: When Map Designer Express performs the print or print-preview task, it copies the attribute transformation table in the Table tab. Before you print, you can adjust the width of the individual columns and height of individual rows in the attribute transformation table to make the whole map fit on one page or to customize the print result.

Deleting objects

This section provides information on how to delete the following objects:

- "Deleting map transformation steps"
- "Deleting business objects"
- "Deleting maps" on page 64

Deleting map transformation steps

Deleting a map transformation step includes three components:

- Deleting the transformation code
- Deleting the comment
- Deleting the data flow arrow

Perform the following steps to delete the transformation step:

- From the Table tab:

  Select the attribute line to delete by clicking in the leftmost column (the column to the left of Exec. Order) and doing one of the following actions:
  - Right-click and select Delete Row from the Context menu.
  - Select the Delete Current Selection option from the Edit menu.
  - Use the keyboard shortcut of `Del`.

  Result: Map Designer Express automatically deletes any incomplete transformations when you save the map.
- From the Diagram tab:

  Select the data flow arrow and select either of the following menu options:
  - The Delete Current Selection option from the Edit menu
  - The keyboard shortcut of `Del`
  - The Delete option from the map workspace's Context menu

  Result: A dialog asks you whether to delete the associated data flow arrow. Click Yes and Map Designer Express displays a second confirmation asking if you want to delete the associated code:

  Click Yes and all three items are deleted.

Deleting business objects

Perform the following steps to delete a business object from a map:

1. Display the Delete Business Object dialog in any of the following ways:
   - Select the Delete Business Object option of the Edit menu.
   - From the Table tab, perform either of the following actions:
     - Right-click in the empty area of the business objects pane and select Delete Business Object from the Context menu.

– Right-click the business object in the business objects pane (click the name in the cell) and select Delete <*BusObjName*> (where *BusObjName* is the name of the selected business object.)

Result: The Delete Business Object dialog displays.

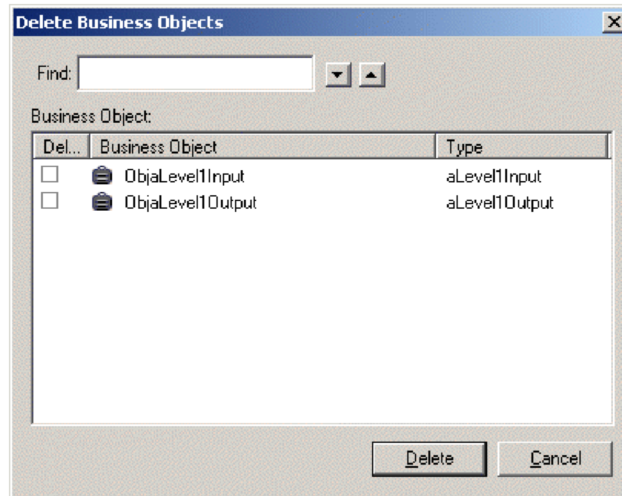


Figure 31. Delete Business Object dialog

2. Through the Delete Business Object dialog, you specify which business objects you want to delete from the map. The Delete Business Object dialog provides the following functionality:
   - To delete a business object:
     – Check the business object in the business object list.
     – Click the Delete button.
   - To locate a particular business object, enter its name in the Find field. The up and down arrows scroll through the business object list.
   - To close the dialog, click Done.

Deleting maps

Perform the following steps to delete a map from the project in System Manager:

1. Select the Delete option from the File menu.

Result: Map Designer Express displays the Delete Map dialog, as Figure 32 shows.

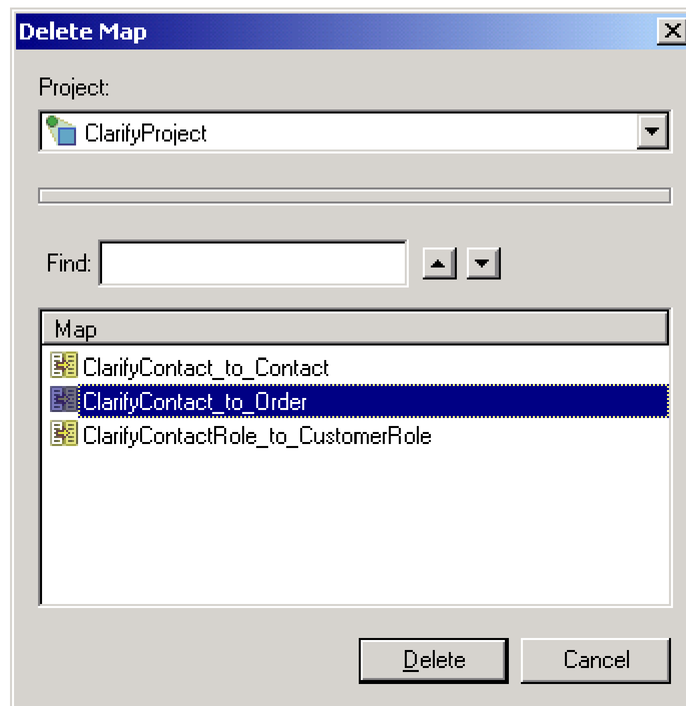


Figure 32. Delete Map dialog

Note: If a map is currently open, Map Designer Express closes this map before it displays the Delete Map dialog. You can specify whether Map Designer Express closes any currently open map with the option `Delete Map: close map before delete`. By default, this option is enabled. If the option is disabled, Map Designer Express provides a confirmation prompt if you select the currently open map from the Delete Map dialog. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.

2. Enter the project name.
3. Select the map or maps you want to delete.

   From the Delete Map dialog, you can:
   - Select a single map by clicking on the map name in the list.
   - Select multiple maps by holding down the `Ctrl` or `Shift` key and clicking on the map names.
   - Locate a particular business object by entering its name in the Find field. The up and down arrows scroll through the business object list.
4. Click the Delete button to delete the maps.

   Result: Map Designer Express displays a confirmation box for the delete.

Note: You can specify whether Map Designer Express confirms the deletion of a map with the option `Delete Map: always display warning message`. By default, this option is enabled. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.

Using execution order

By default, map execution occurs in the order that the destination attributes appear in the Table tab. Only destination attributes that have transformations are executed. Often, the execution order is the order in which the destination attributes are defined in the destination business object. Figure 33 shows an execution order of the map A-to-B in which destination attributes are executed in the order they are defined.

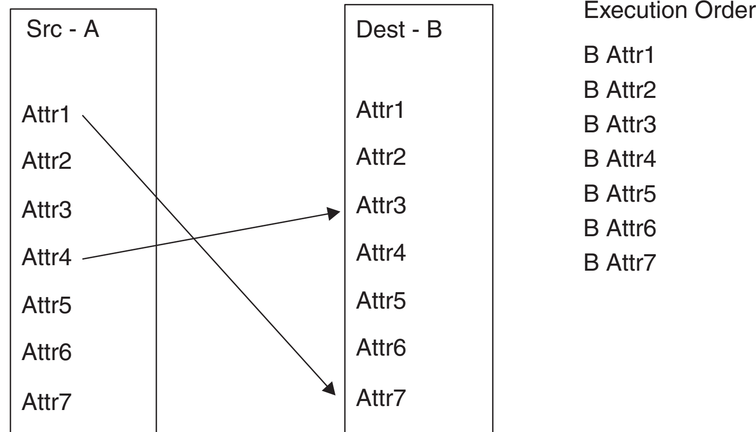


Figure 33. Default execution order

Note: Figure 33 assume that all destination attributes have transformation code.

However, certain attributes might have dependencies in their execution order. To ensure that the transformation code of certain attributes is executed before the transformation code of other ones, you can specify the order of their execution. You can change the execution order to specify data flow. For example, suppose in the map A-to-B that `Attr7` needs to execute immediately after `Attr3` (in other words, `Attr7` needs to execute before `Attr4`). Figure 34 shows how a sequence specification in the destination business operation changes the sequence.

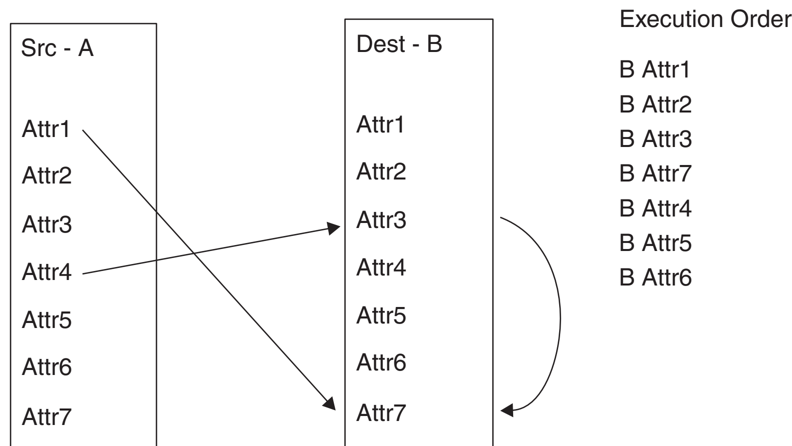


Figure 34. Changing execution order

You can specify an explicit execution sequence that overrides the default order from the Table tab of Map Designer Express. To specify the sequence of

transformations between two destination attributes in the Table tab, click in the Exec. Order field for the destination attribute whose execution order you want to change and enter the desired execution order value.

Note: You can specify whether Map Designer Express renumbers the execution order for any attributes affected by this change with the option `Defining Map: automatically adjust execution order`. By default, this option is disabled. When the option is enabled, Map Designer Express automatically adjusts the execution order of other attributes. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20..

By default, the Table tab displays attributes in the order their transformations are defined. You can then choose to display these mapped attributes by their execution order, their attribute names, or ordered by any other column of the attribute transformation table. Just click the heading of the column to order the attributes by that column's value.

Important: If you click the row header of the transformation and drag-and-drop the transformation to a new position, you change the order in which the transformation rule is displayed. However, this action does *not* affect its execution order.

Importing and exporting maps from InterChange Server Express

With the `repos_copy` utility, you can load and unload specified map definitions in the repository with the `-e` option. A *map repository file* is the file that the `repos_copy` utility creates when it extracts a map definition from the repository into a `.jar` file. This file contains a map definition in an IBM WebSphere InterChange Server Express-defined `.jar` format.

Recommendation: You should use the `.jar` file extension for the map repository file.

For example, the following `repos_copy` command unloads (exports) the `ClCwCustomer` (`ClarifyBusOrg` to generic `Customer`) map definition from the repository of an InterChange Server Express named `dexter` into a map repository file:

```
repos_copy -eMap:ClCwCustomer+BusObj:Customer+BusObj:Clarify_Customer
-oNM_ClCwCustomer.jar -sdexter -pnull -uadmin
```

You can create one repository file that contains all map definition files, including:
- Main map definitions
- Submap definitions
- Files for *both* directions, if applicable.

For example, to copy all related map definitions for the `ClarifyBusOrg/Customer` mapping into a map repository file, use the following `repos_copy` command:

```
repos_copy -eMap:ClCwCustomer+Map:CwClCustomer
-oNM_ClCwCustomer_and_CwClCustomer.jar -sdexter -pnull -uadmin
```

If you are reusing a submap in several maps, create a separate `repos_copy` file for it instead of putting it in the main text file.

You can also use repos_copy to load (import) a map definition into the repository from a map repository file. The following repos_copy command loads the ClCwCustomer map definition into the repository of an InterChange Server Express named testing:

```
repos_copy -iNM_ClCwCustomer.jar -stesting -uadmin -pnull
```

This repos_copy command assumes that the ClCwCustomer and CwClCustomer map definitions do *not* currently exist in the repository. If they do exist, this command fails to load these new map definitions. You can use one of the -a options of repos_copy to choose how to handle duplicate map definitions:

-ai	Skip over duplicate map definitions during the load
-ar	Overwrite any duplicate map definitions with the map definition in the map repository file.
-arp	Interactively query the user whether to overwrite any duplicate map definitions with the map definition in the map repository file.

Note: In Production mode, the maps will be automatically compiled.

You can also use repos_copy to load and unload relationship definitions in the repository. For more information, see "Loading and unloading relationships" on page 221..

Chapter 4. Compiling and testing maps

This chapter describes how to validate, compile, and test maps using Map Designer Express.

- "Validating a map" on page 69
- "Compiling a map" on page 70
- "Compiling a set of maps" on page 71
- "Testing maps" on page 72
- "Debugging maps" on page 85

Validating a map

Map Designer Express's validation process verifies the accuracy of the map's data flow by performing the following checks:

- Ensures that the map has no incomplete transformation steps.
- Ensures that indexes to business object arrays are properly sequenced, starting from zero (0).
- Provides a warning if any transformation step maps to the `ObjectEventId` attribute.
- Validates transformations:
  - Makes sure execution order is correct; that is, that execution order is unique, positive, and consecutive.
  - Ensures that no attributes have cyclic dependencies on each other. If any cyclic transformations are found, Map Designer Express displays the cyclic rules in the output window.
  - Checks transformation information:

    Move transformation—only one source attribute is involved.

    Join transformation—more than one source attribute is involved.

    Split transformation—only one source attribute is involved; split index is greater than or equal to zero; split delimiter is not empty.

    Set Value transformation—no source attribute is involved; a value has been specified.

    Submap transformation—at least one source attribute is involved; submap name is specified.

    Cross-Reference transformation—only one source attribute is involved.

Map Designer Express automatically validates a map when you save it. You can also choose to validate the map by performing either of the following actions:

- Select Validate Map from the File menu.
- In the Designer toolbar, click the Validate button.

At this point, if you have specified any options on the Validation tab of the Preferences dialog, Map Designer Express will issue a warning if the specific condition is not mapped.

For more information on setting dependencies between attributes, see "Using execution order" on page 66.

© Copyright IBM Corp. 2003

Compiling a map

When it compiles a map, Map Designer Express generates a `.class` file from the `.java` file that holds Java code for the map's transformations. It generates this `.java` file from the transformation code stored as part of the map definition in the project.

Important: To be able to compile a map, the Java compiler (`javac`) must exist on your system and its path must be on your `PATH` system variable. For more information, see "Setting up the development environment" on page 10..

From within Map Designer Express, you can initiate compilation of a map in several ways:

- Compile the *current* map in one of the following ways:
  - Select Compile from the File menu.
  - Use the keyboard shortcut of `F7`.
  - In the Designer toolbar, click the Compile button.
- Compile the *current map and any submaps* that this map is using:
  - Select Compile with Submap(s) from the File menu.
- Compile *all* or a subset of maps defined in System Manager:
  - Select Compile All from the File menu.
  - Use the keyboard shortcut of `Ctrl+F7`.

  For more information, see "Compiling a set of maps" on page 71.

By default, Map Designer Express saves the map in the project before it begins the compile and generates the Java code in the `.java` file and `.class` file. If any message file is needed, Map Designer Express will also generate the message file.

Note: You can specify whether Map Designer Express automatically saves a map to the project before compiling the map with the option `Compile Map: save map before compile`. By default, this option is enabled. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.

To compile, Map Designer Express calls the Java compiler on the map's Java source code (`.java` file). The actions it then takes depend upon whether the compilation is successful.

System Manager also provides several ways to compile a map. You can do any of the following:

- Compile a single map:
  - Highlight the desired map and select Compile from the Component menu.
  - Right-click the desired map and select Compile from the Context menu.
- Compile a map and its submaps:
  - Right-click the desired map and select Compile with Submap(s) from the Context menu.
- Compile *all* maps defined in the project:
  - Highlight the Maps folder and select Compile All from the Component menu.

Note: You will need to select which map folder in the project to compile all maps for by right-clicking on the map folder and selecting Compile All from the Context menu.

For more information on using System Manager to compile a map, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

A successful map compilation

When the map successfully compiles, Map Designer Express takes the following steps:

- Compiles the Java code into a `.java` file.
- Displays the following message in the output window at the bottom of each Map tab to indicate that there are no errors during compilation:

  ```
  Compilation is successful.
  ```

An unsuccessful map compilation

If an error occurs during compilation, Map Designer Express generates error messages and displays them in the output window at the bottom of the screen. Unless an output window is already open, Map Designer Express opens one at the bottom of the Map tab to display these compilation messages.

When a compile error occurs, the output window displays the error message with the problematic attribute name and line number in blue. Click the hyperlink to navigate to the problematic area in the Java view in Activity Editor.

Tip: You can clear the output window of messages by choosing Clear Output from the View menu.

Some errors are easy to detect, while others are not.

Compiling a set of maps

Using the Compile All option on the File menu, you can compile all maps in your System Manager, or a subset of maps. Perform the following steps to compile a set of maps:

1. Select Compile All from the File menu.

   Result: Map Designer Express displays the Compile All Maps window.
2. Select the project to compile maps for.
3. Select the maps to compile.

   Guideline: Checking any check box at the root will automatically check all its child check boxes. Thus, when you select a project, all maps in that project are selected. To select only a subset of maps, deselect the appropriate Compile check boxes.

Figure 35 illustrates the Compile All Maps window.

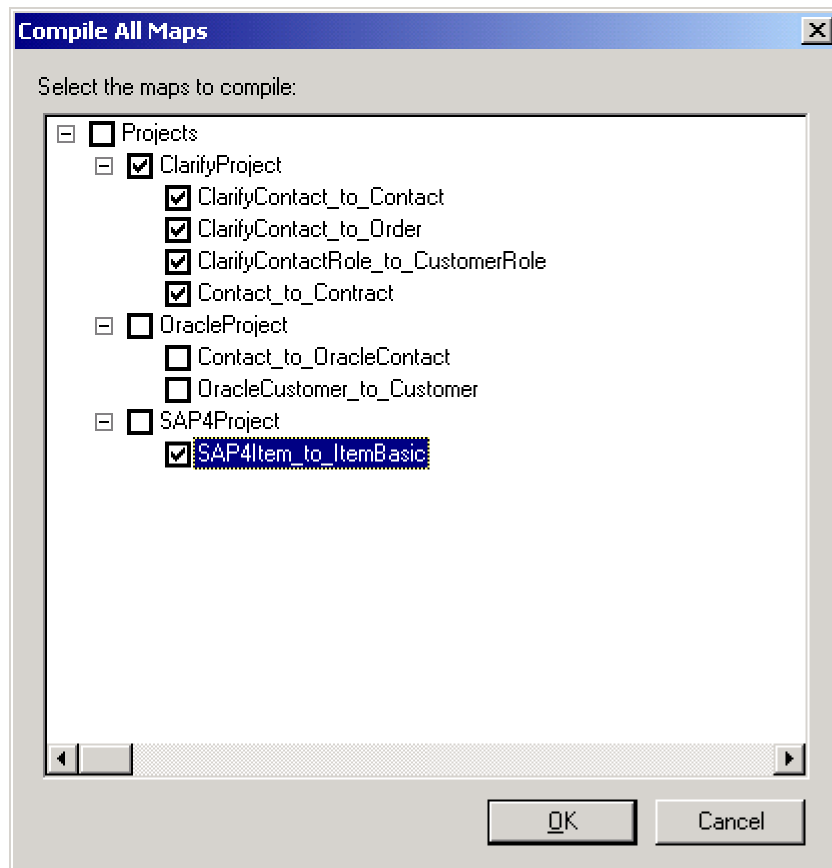


Figure 35. Compile All Maps window

Result: Map Designer Express displays the success or failure of each map's compilation in the output window. You might want to enlarge the size of the output window before starting the compilation process so you can see more of the compilation status messages.

Testing maps

You can test a map's transformation steps by providing sample data for the source business object and executing a test run of the map. A *test run* is map execution that does not involve an event sent by a connector or a call sent by an access client; the map executes within Map Designer Express. Map Designer Express provides a separate tab, the Test tab in the Map Designer Express window to test maps and view test results.

Note: When a map is selected from Testing Environment for further debugging, Testing Environment will launch Map Designer Express, giving Map Designer Express the input business objects to the map under testing.

This section describes how to set up and execute a test run, using these steps:

- "Preparing to run the test" on page 73
- "Creating test data" on page 73
- "Setting breakpoints" on page 75

- “Running the test map” on page 77
- “Viewing test run results” on page 79
- “Changing the map and re-executing” on page 79

Note: An alternative testing strategy, which is not covered in detail, is to set breakpoints in the map and to send a triggering event from the connector, which causes the map to execute.

Preparing to run the test

Before running the test, perform the following steps:

1. Open the map to debug from the project.
2. If the map has *not* been compiled since the last modification, compile it by choosing Compile from the File menu. For more information, see “Compiling a map” on page 70.
3. If the Test tab of Map Designer Express is *not* currently displaying in the tab window, select the Test tab.

Creating test data

Every time you test a map, you must load data into the source business object. To do this, use the Source Testing Data pane in the Test tab (see Figure 36). The Source Testing Data pane allows you to specify the following test information:

- The calling context—indicates the map execution context for the map run.
- The generic business object—provides test data for the generic business object when testing the `SERVICE_CALL_RESPONSE` calling context for an identity relationship.
- The test data—data for the attributes of the source business object.

Important: The calling context and generic business object are required *only* for testing relationships within maps. For more information, see “Testing maps that contain relationships” on page 80.

Testing the map for the first time

When you test the map for the first time, you must manually enter the values of the attributes in the Source Testing Data pane.

The following sections provide information about how to enter this data:

- “Test data for the source business object” on page 73
- “Test data for a child business object” on page 74

Test data for the source business object: To create source business object data for the first time, follow these rules:

- To set the verb, select it from the verb combo box in the verb row.
- To assign a value to a source attribute, type it into the attribute’s Value column. You do *not* have to provide values for all attributes.
- To assign a value to a relationship attribute, specify the appropriate value in the Value column and make sure you also specify the correct calling context. For more information, see “Testing maps that contain relationships” on page 80.
- To assign values to a child business object, right-click the child object and select the Add Instance option from the Context menu. For more information, see “Test data for a child business object” on page 74.
- To assign default values to the source attributes attribute, select the source business object and select Reset from the Context menu.

- If you are testing relationships, make sure to set the `ObjectEventIds` of the source parent object and all child objects that participate in the relationships.

To save the values you have entered for future test runs, create a business object (`.bo`) file by selecting the source business object and performing either of the following actions:

- Click the Save To button in the Source Testing Data pane.
- Select Save To from the Context menu. When prompted, enter a file name where these values will be stored.

  Result: The next time you test this map, you can click the Load From button and the attributes will be filled in automatically from the business object file.

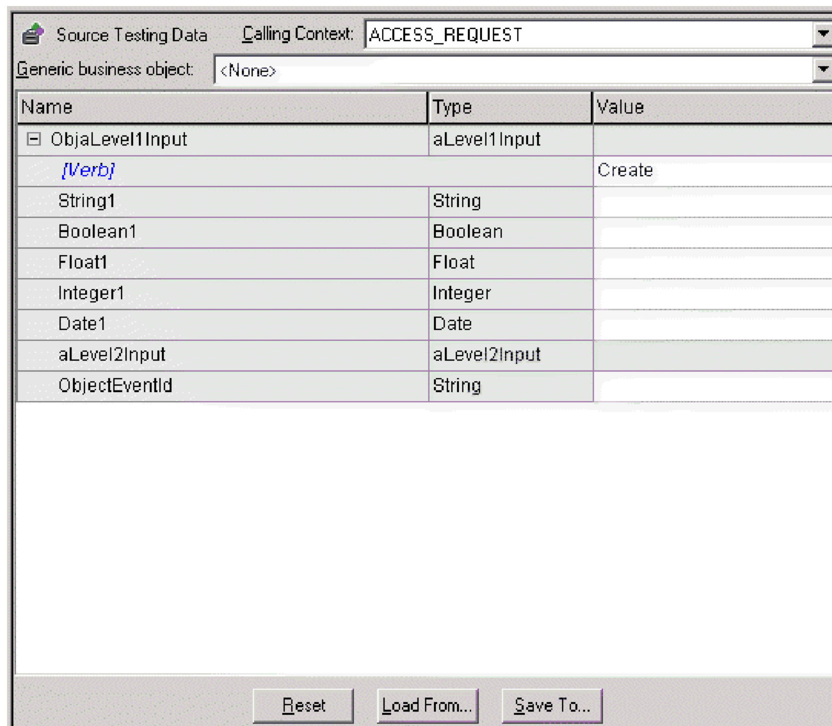


Figure 36. Source Testing Data pane of the Test tab

Test data for a child business object: If the source business object has child business objects and you want to specify test data for the child attributes, you must first create an instance for each child object you need. To do so, perform the following steps:

1. Right-click the child business object name and select Add Instance from the Context menu. When you expand the object, you see the instance that Map Designer Express has created.

   Guideline: The first instance you add has an index number of zero. You can have as many instances as you want (as long as the child attribute has multiple-cardinality).
2. Click the plus symbol (+) beside the instance index number to expand the child business object.

   Result: When you expand the object, you see the child attributes for this instance.
3. To create data for the child business object instance, follow these rules:

- To set the verb for the child business object, select it from the verb combo box in the verb row.
- To specify a value for a child attribute, select it and enter the value in the Value column.
- If the name of the attribute is followed by `(N)`, the attribute contains a multiple-cardinality child business object and you can add more instances.

  To add a child business object to the end of the array, right-click the last index and select Add Instance from the Context menu.
- Modify the values of as many instances as you want. Add and remove instances as follows:
  - To add an instance, right-click the child instance name and select Add Instance.
  - To delete an instance, right-click the instance name of the child instance you want to delete and select Remove Instance.
  - To delete *all* instances, right-click the child instance name and select Remove All Instances. This option is only enabled if the child business object has multiple-cardinality.

Testing the map in subsequent runs

For subsequent test runs, Map Designer Express reuses the previously specified test data. You can take any of the following actions on this data:

- Leave all test data as it is.
- Modify values for any individual attributes by changing the appropriate entries of the Values column.

  Tip: If you modify the data, remember to resave any business object (`.bo`) file.
- Load a set of values from a business object (`.bo`) file.

  To load attribute values from a business object file, select the source business object and perform either of the following actions:
  - Click the Load From button in the Source Testing Data pane.
  - Select Load From from the Context menu.

  When prompted, enter the name of the business object file to be loaded.
- Return all source destination values to their defined default values by selecting the source business object and selecting the Reset option from the Context menu.

Setting breakpoints

When you set a breakpoint, map execution pauses just before the transformation of the destination attribute on which the breakpoint is set. The use of breakpoints lets you step through map execution and check the sequence and the results of individual operations. You can set as many breakpoints as you like.

Guideline: Breakpoints are not part of the map's definition. You set breakpoints on the map after the map is opened in Map Designer Express, and when the map is debugged (either with Debug-->Run Test... or Debug-->Advanced-->Attach...). Breakpoints have no effect on the map when the map is not debugged from Map Designer Express.

Note: You can only set a breakpoint on a destination attribute that has a transformation defined for it.

To set a breakpoint, you can use any one of the following methods:

- Right-click a destination attribute in the Destination Testing Data pane and select Set Breakpoint from the Context menu. If the destination source attribute is not yet expanded, you can expand it with either of the following commands:
  - Click the plus symbol (+) next to the destination business object.
  - Select the destination business object and select Expand from the Context menu.

    Note: The Context menu of the destination business object also provides a Collapse option.
- Select Toggle Breakpoint from the Debug menu.
- Use the keyboard shortcut of `F9`.
- In the Designer toolbar, click the Toggle Breakpoint button.

Note: The Toggle Breakpoint option toggles a breakpoint definition on and off. If the breakpoint is *not* currently set, Toggle Breakpoint sets it. If the breakpoint is currently set, Toggle Breakpoint removes it.

Result: Map Designer Express displays a dark circle next to the destination attribute on which the breakpoint is set, as shown in Figure 37.

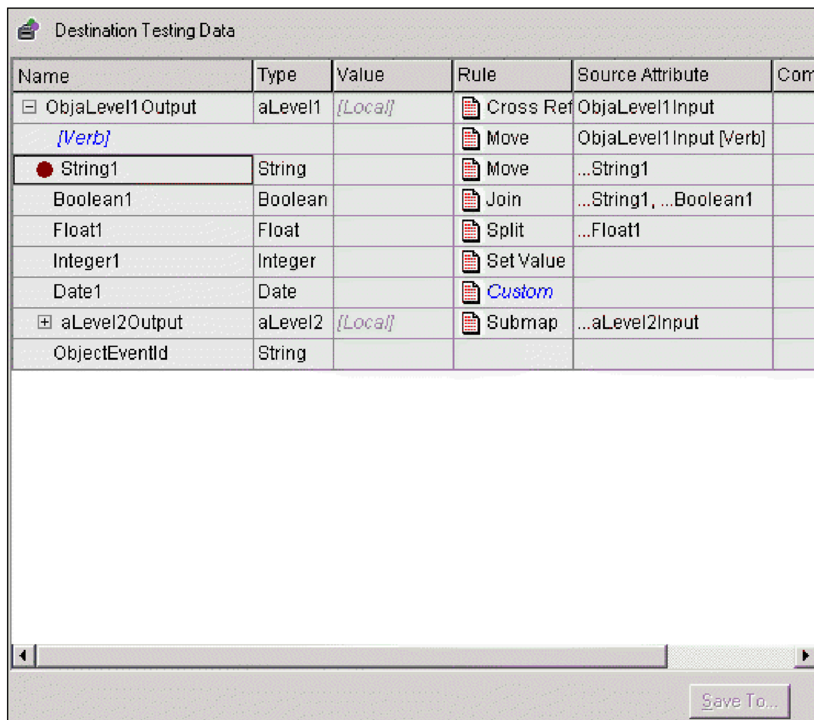


Figure 37. Breakpoint set

Once you set the breakpoint, the execution of the map instance pauses at this breakpoint and you can see the current status of the map. Unless you specify at least one breakpoint, the map executes and finishes with the message:

```
Test run finished
```

Rule: You must always provide values for the source data associated with the destination attributes where you set the breakpoints. Otherwise, the transformation rule will run normally and the breakpoints will execute normally, but the

destination value will usually be empty, depending on what transformation rule is defined. For more information, see "Creating test data" on page 73.

To view all breakpoints for the map, select Breakpoints from the Debug menu.

Result: Map Designer Express displays the Breakpoints dialog (see Figure 38).

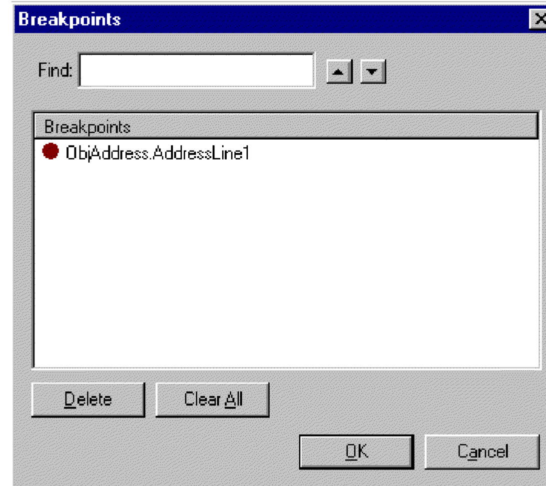


Figure 38. Breakpoints dialog of the test tab

From the Breakpoints dialog, you can perform any of the following actions:

- Locate a destination attribute on which a breakpoint is set—double-click the breakpoint name.

  Tip: To locate a particular breakpoint, enter its name in the Find field. The up and down arrows scroll through the business object list. In the Destination Testing Data pane, Map Designer Express highlights the destination attribute.
- Remove a breakpoint—in the Breakpoints area, select the breakpoint to remove and click the Delete button.

  You can also remove a breakpoint by performing any of the following actions:
  - Right-click a destination attribute in the Destination Testing Data pane and select Clear Breakpoint from the Context menu.
  - Use any of the commands for the Toggle Breakpoint option on an existing breakpoint. For more information, see "Setting breakpoints" on page 75.
- Clear all breakpoints that display in the Breakpoints area—click the Clear All button.

  You can also clear all breakpoints by performing any of the following actions:
  - Select Clear All Breakpoints from the Debug menu.
  - In the Designer toolbar, click the Clear All Breakpoints button.

Running the test map

Once you have entered the source test data and set any desired breakpoints, you are ready to test the map. To run a map test involves the following steps:

1. "Starting the test run" on page 78
2. "Processing breakpoints" on page 78 (if any breakpoints have been set)

Starting the test run

To start the test run, perform the following steps:

1. Do any of the following actions:
   - Select Run Test from the Debug menu.
   - In the Designer toolbar, click the Run Test button.

   Result: The Connect to IBM WebSphere InterChange Server Express dialog box will display and allow you to connect to the server for testing.
2. In the dialog, enter the server name, user name, and password.
3. Specify whether you want to deploy the map and dependent business objects for the test run.

   Guideline: Deploying a minimum set of business objects to the server for testing will minimize debugging initialization time.

Result: Execution of the map starts. Map Designer Express displays the following message in the output window:

```
Starting test run...
```

Processing breakpoints

Map execution pauses when it reaches a destination attribute where you have set a breakpoint. When the breakpoint is reached, Map Designer Express takes the following actions:

1. Highlights the destination attribute on which the breakpoint was set and displays a dark circle with a yellow arrow next to it.
2. Displays the following message in the output window:

   `Test Run stopped at attribute` *`AttrName`* `(next transformation--> "`*`Rule`*`").`

   Tip: With map execution paused, you can examine the values of the destination attributes that have been processed so far by looking in the Value column of the Destination Testing Data pane.
3. Processes the breakpoint and continues map execution, when you do either of the following actions:
   - Proceed to the next breakpoint or the end of the map, whichever comes first.

     To continue map execution, perform any of the following actions:
     - Select Continue from the Debug menu.
     - Use the keyboard shortcut of `F8`.
     - In the Designer toolbar, click the Continue button.
   - Execute this destination attribute, then stop before executing the next attribute.

     To continue map execution for only one more step, perform any of the following actions:
     - Select Step Over from the Debug menu.

       Tip: Select this option to watch the code execute attribute by attribute.
     - Use the keyboard shortcut of `F10`.
     - In the Designer toolbar, click the Step Over button.

   Result: When the execution of the test run is finished without any run-time errors, Map Designer Express displays the following message in the output window:

   ```
   Test run finished.
   ```

Viewing test run results

Test run results display in the destination business object, which is in the Destination Testing Data pane. Values resulting from the map transformations are visible in the Values column of this table. You can view test run results by either:

- "Watching the process"
- "Viewing results after execution"

Watching the process

During a test run that has test data and breakpoints, you can watch as the destination business object fills with values. Values appear in the Values column in the Destination Testing Data pane as they are processed. When map execution is paused on a breakpoint, all destination attributes *before* that attribute in the execution order have values displayed.

To view the transformations as they occur:

- Set a breakpoint on the second destination attribute and step through map execution with the Step Over option. The map will be read-only.

Viewing results after execution

To view test run results when the map has already executed, examine the destination business object in the Destination Testing Data pane.

To save the test results:

- Highlight the destination business object and select Save To from the Context menu.

  Result: Map Designer Express saves the values of the destination attributes in a business object (`.bo`) file.

Changing the map and re-executing

As you test the map, you might discover the need to change the map. To edit the map and then continue the test, perform the following steps:

1. Switch to either the Table or Diagram tab to view the map transformations.
2. Make the edits to fix the errors.
3. Recompile the map.
4. Continue the testing process by switching back to the Test tab.
5. Begin a new test run.

Important:

1. Make sure you complete the test run, either with success or failure, before you attempt to recompile the map.
2. After you modify the map, be sure to deploy the map to the server for the change to be reflected in the server.

Doing advanced debugging

Besides debugging maps that are stored in local projects, you can also directly debug a map that resides in the server. Perform the following steps to do so:

1. Select Debug-->Advanced-->Attach.

   Result: The Connect to WebSphere InterChange Server Express dialog displays.
2. Enter the Server name, User name, and Password; and click Connect.

   Result: Map Designer Express displays a list of new maps on that server.
3. Select the map you want to attach to.

Result: The map opens in Map Designer Express in Read-only mode.

4. Set breakpoints in the map to have the server pause map execution at a certain transformation rule.

   Result: When a breakpoint is hit on the server, you can step over or continue map execution, as usual. The resulting business object values will display in the Destination Test Data pane.
5. Stop the debugging session at any time using Debug-->Advanced-->Detach.

   Result: Map Designer Express will close the map.

Testing maps that contain relationships

When you test a map that contains a relationship transformation, you need to provide the following information in addition to the test data:

- The calling context

  Part of a map's execution context includes a calling context. Many of the relationship methods in the Mapping API use this calling context to determine what action to take during the mapping. For this reason, if you are testing a relationship attribute in a map, you usually must specify the appropriate calling context for the transformation.
- The generic business object definition

  When you test the `SERVICE_CALL_RESPONSE` calling context for an identity relationship, you need to specify the maps generic business object so that the test run can locate the generic key value in the relationship.

You specify this information in the Source Testing Data pane of the Test tab.

Tip: If the width of the Source Testing Data pane is not enough to let you see the complete menu options of the Calling Context combo box, you can expand the size of this area by putting the cursor over the right-hand boundary until you see the following symbol `<-||->` and drag the boundary to the right.

If you are testing Relationships, select the appropriate generic object from the list of business objects, select Calling Context, and set the `ObjectEventIds` for the parent and child objects that match the ones you already set in the Test Data screen. The calling context you need to provide and whether you need to specify a generic business object depend on the type of relationship you are testing. This section provides information on the following:

- "Testing an identity relationship"
- "Testing a lookup relationship" on page 83

Testing an identity relationship

To test point-to-point mapping (from Application 1 to Application 2) for an identity relationship you use three maps:

- An inbound map from Application 1's application-specific business object to a generic business object—`App1_to_Generic`
- An outbound map from the generic business object to Application 2's application-specific business object—`Generic_to_App2`
- An inbound map from Application 2's application-specific business object to the generic business object—`App2_to_Generic`

Figure 39 shows an example of a point-to-point communication of customer data between a Clarify application and an SAP application. If each application uses a

unique key value to identify customers, these three business objects can be related with an identity relationship. Therefore, each map includes a cross-reference transformation rule. As each of these maps executes, these relationship methods access the calling context to determine the actions to take.

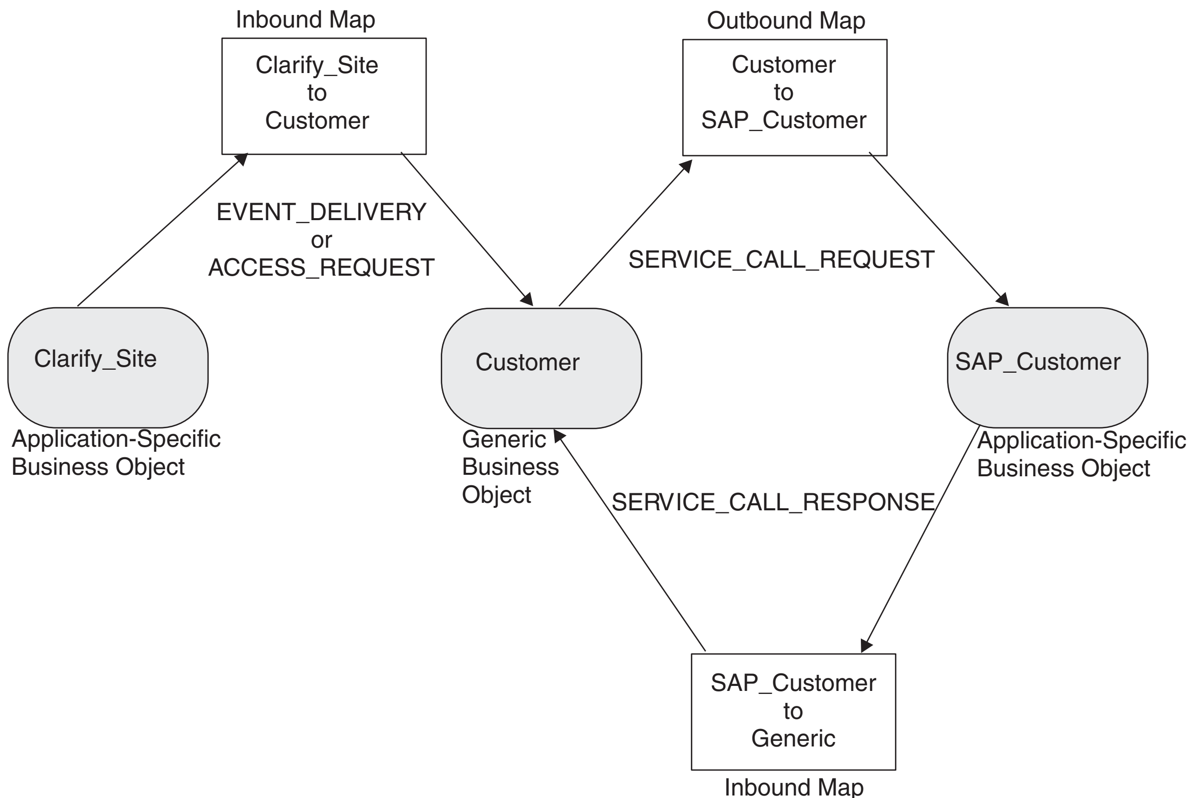


Figure 39. Maps involved in point-to-point testing of an identity relationship

To test the Create verb, you need to verify that a new application-specific key value in Application 1 (Clarify application in Figure 39) causes a new generic key value to be added for the generic business object *and* a new application-specific key value in Application 2 (SAP application in Figure 39). Therefore, testing involves three steps:

1. Test the inbound map, `App1_to_Generic`, to send in a new key value from Application 1 and ensure that a new key value is generated for the generic business object. Follow the steps in Table 17.

Table 17. Testing the App1-to-Generic map for an identity relationship

To set up test run	To verify test run
1. Set the calling context to `EVENT_DELIVERY` or `ACCESS_REQUEST` by choosing the appropriate calling context from the Calling Context combo box. 2. Enter the application-specific value in the key of the source business object. This value is unique for the key attribute(s) in the `Application1` application. 3. Run the test.	4. Read the resulting generic key value in the destination business object, which has been added to the relationship table for the `App1/Generic` identity relationship. 5. Save the destination business object data in a `.bo` file (e.g. `App1_to_Generic.bo`) by selecting the destination business object and choosing Save To from the Context menu.

2. Test the outbound map, Generic_to_App2, to ensure that the new generic key value is sent to Application 2.

   To test an identify relationship in the outbound Generic_to_App2 map, you must provide the generic key value in your source Test Data. You might want to do either of the following, *but they are both wrong*:

   - Put an arbitrary number into the generic business object's primary key attribute, then run the map.
   - Create the record directly in the relationship table.

   In both cases, Map Designer Express generates the RelationshipRuntimeException or NullPointerException. The error occurs because the generic key value has to be in the system for the SERVICE_CALL_REQUEST to work properly, and the relationship table is *not* the only place the generic key value is stored.

   The correct solution is to first run an inbound EVENT_DELIVERY (or ACCESS_REQUEST) map that uses the same identity relationship (as described in step 1). Follow the steps in Table 18 to test the outbound Generic_to_App2 map.

Table 18. Testing the generic-to-app2 map for an identity relationship

To set up test run	To verify test run
1. Set the calling context to SERVICE_CALL_REQUEST by choosing this calling context from the Calling Context combo box. 2. Load the generic business object with the test results from the previous step (e.g. App1_to_Generic.bo). 3. Run the test.	4. Read the resulting application-specific key value in the destination business object, which is empty because Application 2 has not generated its key value yet. 5. Save the destination business object data in a .bo file (e.g. Generic_to_App2.bo) by selecting the destination business object and choosing Save To from the Context menu.

3. Test the inbound map, app2_to_generic, to verify that the new key value from Application 2 is associated with the new generic key value.

   When the calling context is SERVICE_CALL_RESPONSE, an identity relationship must cross-reference the ID in the application-specific business object to the ID in the generic business object. Therefore, for this test, you must specify the generic business object definition. Follow the steps in Table 19.

Table 19. Testing the App2_to_Generic map for an identity relationship

To set up test run	To verify test run
1. Set the calling context to `SERVICE_CALL_RESPONSE` by choosing this calling context from the Calling Context combo box. 2. Set the generic business object by choosing the name of the appropriate generic business object from the Generic Business Object combo box. Map Designer Express adds the specified generic business object to the Source Testing Data pane. 3. Load the application-specific business object with the test results from the previous step (e.g. `Generic_to_App2.bo`). 4. In the application-specific business object, enter an application-specific value in the key of the business object. 5. In the generic business object, enter the generic key value associated with the `Application1` key. This value should be the same key value generated for the generic business object in the `EVENT_DELIVERY/ACCESS_REQUEST` test (step 1). 6. Run the test.	7. Read the resulting generic key value in the destination business object, which should be the same value you entered in the generic source business object. 8. You can use Relationship Manager to verify that the correct application-specific key values are associated with this generic key value for this identity relationship. For more information on Relationship Manager, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Testing for other verbs involves similar steps. For more detailed information on the actions of relationship methods for an identity relationship, see Chapter 8, “Implementing relationships,” on page 187.

Testing a lookup relationship

To test point-to-point mapping (from Application 1 to Application 2) for a lookup relationship you use two maps:

- From Application 1’s application-specific business object to a generic business object—`App1_to_Generic`
- From the generic business object to Application 2’s application-specific business object—`Generic_to_App2`

Figure 40 shows an example of a point-to-point communication of customer data between a Clarify application and an SAP application. If each application uses a special static code to identify geographic states, these three business objects can be related with a lookup relationship. Therefore, each map includes custom transformations that do static lookups. For more information, see the ″Static Lookup″ activity example in “Example 3 of using the Activity Editor” on page 133. As each of these maps executes, these relationship methods access the calling context to determine the actions to take.

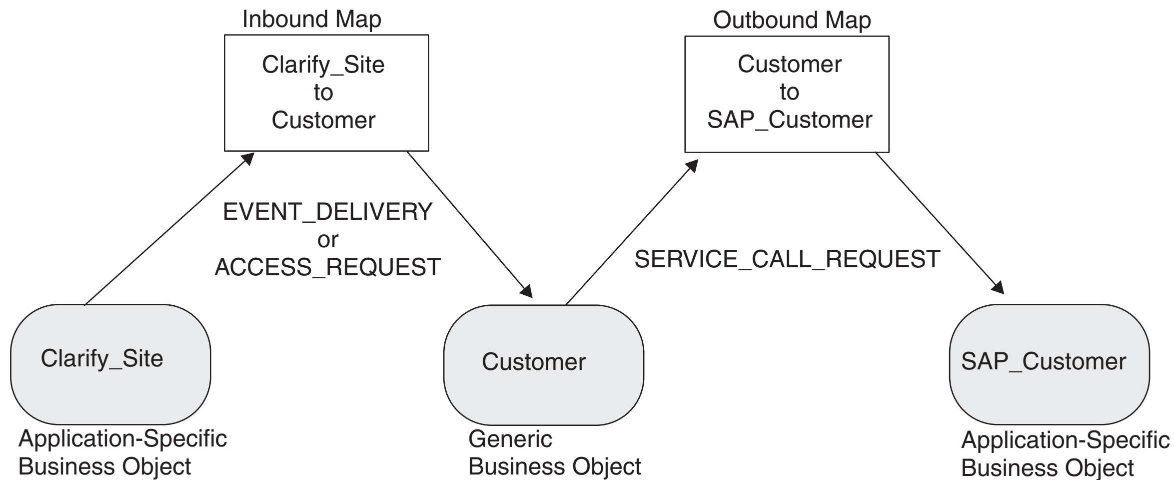


Figure 40. Maps involved in point-to-point testing of a lookup relationship

To test the Create verb, you need to verify that an existing application-specific lookup value in Application 1 (Clarify application in Figure 40) causes the associated generic lookup value to be added to the generic business object *and* the associated application-specific lookup value in Application 2 (SAP application in Figure 40) to be added to its business object. Therefore, testing involves two steps:

1. Test the inbound map, `App1_to_Generic`, to send in an existing lookup value from Application1 and ensure that the associated generic lookup value is obtained for the generic business object. Follow the steps in Table 20.

Table 20. Testing the App1-to-Generic map for a lookup relationship

To set up test run	To verify test run
1. Set the calling context to `EVENT_DELIVERY` or `ACCESS_REQUEST` by choosing the appropriate calling context from the Calling Context combo box. 2. Enter the application-specific value in the lookup field of the source business object. This value is an existing lookup value whose data is already loaded in the `App1/Generic` relationship table. 3. Run the test.	4. Read the resulting generic lookup value in the destination business object, which has been obtained to the relationship table for the `App1/Generic` lookup relationship. 5. Save the business object data in a `.bo` file (e.g. `App1_to_Generic.bo`) by highlighting the destination business object and choosing Save To from the Context menu.

2. Test the outbound map, `Generic_to_App2`, to send in the generic lookup value and ensure that the associated lookup value is obtained for Application 2. Follow the steps in Table 21.,

Table 21. Testing the Generic-to-App2 Map for a lookup relationship

To set up test run	To verify test run
1. Set the calling context to `SERVICE_CALL_REQUEST` by choosing this calling context from the Calling Context combo box. 2. Load the generic business object with the test results from the previous step (e.g. `App1_to_Generic.bo`). 3. Run the test.	4. Read the resulting application-specific key value in the destination business object, which contains the Application 2 lookup value. 5. Save the business object data in a `.bo` file (e.g. `Generic_to_App2.bo`) by highlighting the destination business object and choosing Save To from the Context menu.

Note: A lookup relationship can be tested for the SERVICE_CALL_RESPONSE calling context. However, this case usually only is required if the map is doing something else that requires the lookup data. The relationship methods for a lookup relationship in the Mapping API never write data to a relationship table.

Debugging maps

This section provides the following information about debugging a map:

- "Resolving run-time errors"
- "Debugging tips"

For information on how to test relationships, see "Testing maps that contain relationships" on page 80.

Resolving run-time errors

Even if your map compiled successfully, you can get a run-time error during the map execution in the Debugger.

Example: You have an outbound map with the generic business object on one side and an application specific business object on the other side. Let us assume that this map has an identity relationship in it.

1. Go to the Test tab and select the calling context SERVICE_CALL_REQUEST.
2. Select the verb ″Update.″
3. Run the test.

   Result: An error message like the one below displays:

   ```
   Exception at step 17, attribute <attribute name>,java.lang.nullpointerexception
   ```

This exception is happening because the map is trying to update an entry in the repository that is not created in the first place. Ideally, you should ensure that the sequence of steps is correct. You should look at the database for relationship entries pertaining to the map in question. You should then draw the conclusions based on whether it is ready for SERVICE_CALL_REQUEST or not.

Debugging tips

This section provides the following tips for making the debugging of a map easier:

- "Using logging messages"
- "Writing safe mapping code" on page 86

Using logging messages

Use the logInfo() method for tracking the map execution. It takes a String as an argument, which is sent on the InterChange Server Express log. You need to type it in Activity Editor for the attribute whose execution needs to be tracked. To make sure that the submap is executed, create a custom transformation rule and use the ″Log Information″ function block to customize the activity.

You might not always want to see this message. If this is the case, change the DataValidationLevel property of the map.

To set the DataValidationLevel, select the Map Properties option from the Edit menu of Map Designer Express and change 0 to 1 or a greater number. The settings are as follows:

0	No data validation
1	IBM data validation level
2 or greater	User-defined data validation

Writing safe mapping code

If you customize your transformation rule in Activity Editor, you are *not* guaranteed that it will work properly during run time. To make sure that the map continues executing when an error occurs and you get a notification of an error, use the "Catch Error" function block in Activity Editor and handle the error appropriately.

Chapter 5. Customizing a map

This chapter provides information to use for customizing maps.

This chapter covers the following topics:

- "Customizing transformation steps" on page 87
- "Importing Java packages to Interchange Server Express" on page 136
- "Using variables" on page 140
- "Reusing map instances" on page 144
- "Handling exceptions" on page 144
- "Creating custom data validation levels" on page 146
- "Understanding map execution contexts" on page 146

Customizing transformation steps

Map Designer Express provides two ways to generate Java code:

- Using the Activity Editor
- Defining transformation rules using standard transformations

Using the Activity Editor

Using the Activity Editor, you can specify the flow of activities for a specific transformation rule graphically, without knowing programming or Java code. For each transformation rule in Map Designer Express, you can display one activity and its subactivities. You can view the associated attribute's transformation code graphically, modify it, and have the tool generate the corresponding Java code.

You launch the Activity Editor directly from Map Designer Express (see "Starting the Activity Editor" on page 87). At startup, the Activity Editor communicates with System Manager to discover the set of activities allowed. After you have finished designing the activity for a particular transformation rule, you save the changes in the Activity Editor, and they are communicated to Map Designer Express.

This section contains the following topics on the Activity Editor:

- "Starting the Activity Editor" on page 87
- "Layout of the Activity Editor" on page 88
- "Using the Activity Editor functionality" on page 88
- "Working in Graphical view" on page 91
- "Identifying supported function blocks" on page 95
- "Example 1 of using the Activity Editor" on page 122
- "Example 2 of using the Activity Editor" on page 126
- "Working in Java view" on page 135

Starting the Activity Editor

You launch the Activity Editor through the transformation rule column of the Table or Diagram tabs of Map Designer Express. There are two ways to access information in this transformation rule column:

- Double-click the attribute's corresponding cell of the transformation rule column.

© Copyright IBM Corp. 2003

- Click the bitmap icon in the corresponding cell of the transformation rule column.

Transformation code is generated from one of the standard transformations that Map Designer Express provides on the combo box of the transformation rule column. When you double-click the attribute's transformation rule cell or click the mapping rule icon, the type of transformation determines what Map Designer Express displays:

- For the Custom transformation, Map Designer Express brings up the Activity Editor on the transformation code.
- For all other standard transformations (Join, Set Value, Split, and Submap, Cross-Reference), Map Designer Express displays the transformation's dialog. Click the View Code... button on this dialog to bring up the Activity Editor.

The Activity Editor appears with the attribute name in the title bar. You can open multiple instances of the Activity Editor at the same time.

Layout of the Activity Editor

The Activity Editor has two main views: Graphical view and Java view. Depending on the nature of the activity, at any given time, only one view is visible. Thus, if Map Designer Express invokes the Activity Editor to display a graphical activity, the Activity Editor will startup with the Graphical view. If you choose to translate this graphical activity into Java code, the Java view will display in place of the Graphical view.

Figure 44 on page 92 and Figure 68 on page 136 show the layout of the Graphical and Java views, respectively, of the Activity Editor.

Both views have common Window elements, as described in Table 22..

Table 22. Common Window elements

Window element	Description
Title Bar	Contains the name of the application (Activity Editor), application icon, and the main activity's name.
Menu	Contains the primary menus.
Tool Bar	Contains dockable toolbars with shortcuts to various functions and tools.
Document Display Area	Displays the representation of the activity definition. It is organized with a workbook look.
Status Bar	Displays status information and some handy shortcuts.

Using the Activity Editor functionality

You can access the Activity Editor's functionality using any of the following:

- The pull-down menus at the top of the window
- The Context menu
- Keyboard shortcuts
- The icons in the toolbars

Main menus and keyboard shortcuts: The Activity Editor provides the following pull-down menus:

- File menu

- Edit menu
- View menu
- Tools menu
- Help menu

The following sections describe the options of each of these menus and their associated keyboard shortcuts.

Functions of the File menu: The File pull-down menu of the Activity Editor provides the following options:

- Save [Ctrl+S]--Saves the activity to Map Designer Express.
- Print Setup... [Ctrl+Shift+P]--Brings up the Print Setup dialog box for changing the printer and printing options.
- Print Preview--Switches the editor to print preview mode.
- Print... [Ctrl+P]--Brings up the Print dialog box for printing the current activity.
- Close --Closes the Activity Editor.

Functions of the Edit menu: The Edit pull-down menu of the Activity Editor provides the following options:

- Cut [Ctrl+X]--Deletes the selected item and copies it to the clipboard.
- Copy [Ctrl+C]--Copies the selected item to the clipboard.
- Paste [Ctrl+P]--Pastes the object in the clipboard to the cursor position if they are compatible.
- Delete [Del]--Deletes the selected item.
- Select All [Ctrl+A]--Selects all items.
- Find... [Crtl+F]--Finds the specific text in the editing area.
- Goto Line... [Ctrl+G]--Goes to a specific line.

Functions of the View menu: The View pull-down menu of the Activity Editor provides the following options:

- Design mode--Toggles between Design mode and Quick view mode. (Only one mode is enabled at a single time.)
- Quick view mode--Toggles between Quick view mode and Design mode. (Only one mode is enabled at a single time.)
- Go To--A submenu that provides the following options:
  – Back [Alt+Left Arrow]--Goes backward in the navigation history in Graphical view.
  – Forward [Alt+Right Arrow]--Goes forward in the navigation history in Graphical view.
  – Up One Level--Shows the diagram at one level up.
  – Home [Alt+Home]--Goes to the top-level diagram in Graphical view.
- Zoom In [Ctrl++]--Magnifies content in the editor.
- Zoom Out [Ctrl+-]--Shrinks content in the editor.
- Zoom To... [Crtl+M]--Displays the Zoom dialog box for a zoom factor.
- Library window--Toggles the Library window on and off.
- Content window--Toggles the Content window on and off.
- Properties window--Toggles the Properties window on and off.
- Toolbars--A submenu that provides toolbars (Standard, Graphics, and Java) that toggle on and off.

- Status Bar--Toggles the status bar on and off.
- Preferences... {Ctrl+U]--Opens the Preferences dialog box for changing the default behavior of the editor.

Functions of the Tools menu: The Tools pull-down menu of the Activity Editor provides the following option:

- Translate [Ctrl+T]--Translates the current activity to Java code and brings up the Java view.

Functions of the Help menu: The Help pull-down menu of the Activity Editor provides the following options:

- Help Topics [F1]--Opens the context-sensitive Help topics
- Documentation--Opens the IBM WebSphere InterChange Server Express documentation.

Context menu: The Context menu provides options for performing many tasks on the editing canvas. To access the Context menu, right-click the editing canvas. The Context menu provides the following options:

- New Constant--Creates a new Constant container on the canvas.
- Add Label--Creates a new label component on the canvas.
- Add Description--Creates a new description component on the canvas.
- Add Comment--Creates a new comment component on the canvas.
- Add To do--Creates a new component for entering some reminder in the activity.
- Add To My Collection--Creates a new group in the Library window for reuse.

Toolbar elements: The toolbars provide direct access to various features and functions of the Activity Editor. The functions of the toolbar buttons are the same as their corresponding menu items.

The Activity Editor supports two toolbars:

- Standard toolbar
- Graphics toolbar

Tip: To identify the function of each toolbar button, roll over each button with your mouse cursor.

Standard toolbar: Figure 41 shows the Standard toolbar.

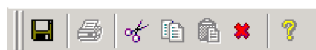


Figure 41. Activity Editor Standard toolbar

Table 23 provides the function of each Standard toolbar button (left to right) and the corresponding menu command.

Table 23. Functions of Standard toolbar buttons

Function	Corresponding menu command
Save Activity	File-->Save
Print Activity	File-->Print...
Cut	Edit-->Cut
Copy	Edit-->Copy

Function	Corresponding menu command
Paste	Edit-->Paste
Delete	Edit-->Delete
Help	Help-->Help Topics

Table 23. Functions of Standard toolbar buttons (continued)

Graphics toolbar: Figure 42 shows the Graphics toolbar.

Figure 42. Activity Editor Graphics toolbar

Table 24 provides the function of each Graphics toolbar button (left to right) and the corresponding menu command.

Table 24. Functions of Graphics toolbar buttons

Function	Corresponding menu command
Back	View-->Go To-->Back
Forward	View-->Go To-->Forward
Up One Level	View-->Go To-->Up One Level
Home	View-->Go To-->Home
Zoom In	View-->Zoom In
Zoom Out	View-->Zoom Out

Status bar elements: The Activity Editor also provides a Status bar, as shown in Figure 43.

Figure 43. Activity Editor Status bar

Table 25 describes the functionality of each Status bar element, left to right.

Table 25. Functions of Status bar elements

Element	Function
Zoom: 100%	Edit box for specifying a zooming percentage
Ready	Status message
10.9	Navigation pane showing the current position of the *I-bar* in the Java editor
>> (Shown in Quick view mode)	Toggle between Design mode and quick view mode
<< (Shown in Design mode)	

Working in Graphical view

If Map Designer Express opens the Activity Editor with an activity definition that has a graphical nature, the Activity Editor will display the activity definition in graphical view in one of two available display modes: Design mode or Quick view mode.

- **Design mode:** In Design mode, the Activity Editor resembles a regular application--in addition to the editing area, it contains a menu bar, toolbars, and other control bars that support your editing needs during the design stage of the activity definition.

  Figure 44 shows the Graphical view in Design mode.

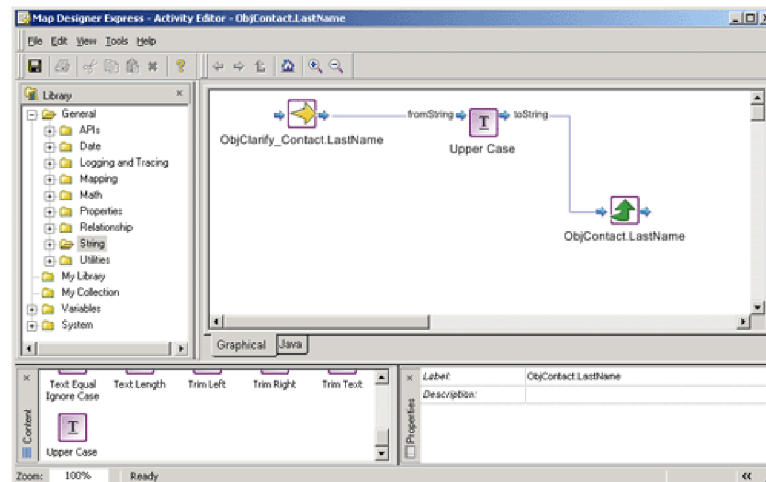


Figure 44. Graphical view in Design mode

  This view contains a main activity editing area (the activity workbook window) and three supporting windows, as follows:

  – Library window---A dockable control bar containing a tree view of the available function blocks, and optionally, the named groups. The function blocks are arranged in folders according to their purpose, and you can expand them to show the actual function blocks. You can view the function blocks in the Library window under their corresponding folder, or as icons in the Content window.

    Additionally, the Library window contains folders for adding system elements to the graphical canvas (System folder), for customizing the library (Library folder), for grouping components (My Collection folder), and for listing global variables accessible to the current activity--typically, the source and destination business object, and the global variable `cwExecCtx` (Variables folder).

  – Content window--A dockable control bar containing a large icon list of the available function blocks under the currently selected folder in the Library window. You can select a function block to view its description and properties in the Properties window, or drag-and-drop a function block onto the editing canvas to create part of the activity flow.

  – Properties window--A dockable control bar containing the properties of the selected component in a gridlike layout. Different components may have different properties. Your interaction with the grid depends on the nature of the individual property. Some properties may be editable; some may be read-only, while some properties may present a drop-down combo-box for your selection. In each case, the Properties window presents the property with appropriate actions.

- **Quick view mode:** In Quick view mode, the Activity Editor resembles a control bar--with only the editing area displayed; all other supporting windows, the menu bar, and the toolbars are hidden.

Figure 45 shows the Graphical view in Quick view mode.

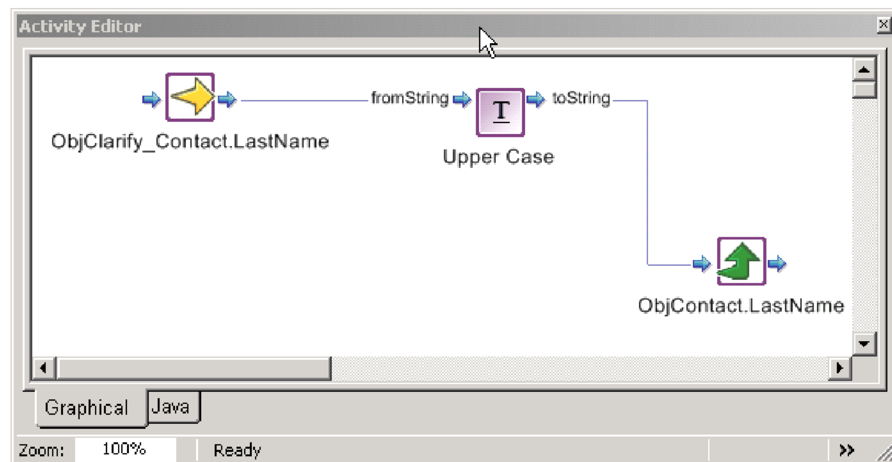


Figure 45. Graphical view in Quick view mode

Initially, when an activity definition that has a graphical nature opens, the Activity Editor displays the top-level view of the definition in a tabbed window. Inside the tab window is the *editing canvas*, which is also known as the *activity canvas* or *graphical canvas*.

Working with activity definitions: You define and modify activity definitions on the editing canvas using the canvas components. The following list identifies the canvas components and briefly describes how to use them to define and modify activity definitions. For detailed steps for defining and modifying activity definitions, see “Example 1 of using the Activity Editor” on page 122, “Example 2 of using the Activity Editor” on page 126, and “Example 3 of using the Activity Editor” on page 133.

- *Function blocks*--define the behavior of an activity. You drag and drop them from the Library window or Content window onto the editing canvas. Each function block has a set of predefined inputs and outputs.

  Result: The Activity Editor displays a little icon for each of the input and output going into or coming out of the block. These *ports* serve as connecting points for linking between the function block and other components. Outgoing ports can connect to multiple connection links, but incoming ports can only connect to one connection link. The name of the input and output displays beside the connection ports. You can choose to show or hide these port names using the option in the Preference dialog.

  For a list of supported functional blocks, organized in tables according to category, see “Identifying supported function blocks” on page 95.

  Note: In addition to the standard function blocks that Activity Editor provides, you can import your own Java library for use as function blocks in Activity Editor. Importing custom Jar libraries into activity settings will enable any public methods in the Jar library to be used as function blocks in Activity Editor. For more information, see “Importing Java packages to Interchange Server Express” on page 136.

- *Connection links*--define the flow of activity between various components in the canvas.

  Example: To specify that the output of function block A should go to the input of function block B, click and hold down the left mouse button on the outgoing

port of function block A, and while continuing to hold down the left mouse button, move the cursor onto the incoming port of function block B, and release the left mouse button. This will create a connection link from function block A's out-port to function block B's in-port. If function block B's in-port is already connected with another connection link, the newer connection link will replace the existing connection link. Graphically, the connection link will appear as a right-angled line between components.

- *Label*, *Description*, *Comment*, and *To Do* tags--identify each activity or subactivity or serve as some reminder in the activity:
  - To start editing them, single click around the center.

    Result: The cursor will change to an I-beam.

    Type the text. All the editing components will wrap the line if the line is longer than the display area. If you want to start a new line, press Enter.
  - To resize the text input field, hold down the left mouse button in the lower right-hand corner of the tag.

    Result: The cursor will change to the resize cursor.

    Move the cursor to resize the editing pad.

    Restriction: Each of these editing components has a minimize size, so the components cannot be resized to be smaller than a certain size.

    Figure 46 shows resizing a label tag and entering multiple lines of text.

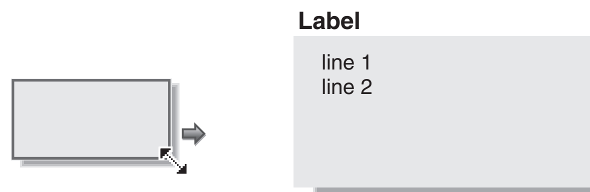


Figure 46. Resizing a label and entering multiple lines of text

  - To move the tag around the canvas, click the edge of the component to drag-and-drop it.
- *New Constant icon*--defines a constant value that you set and use as input to function blocks or ports. When you drag-and-drop the New Constant icon from the Library window or Content window onto the editing canvas, the Activity Editor displays a Constant icon as the container for the constant value. A text edit box displays on top of the icon for you to enter the value of the Constant. To revise this value, double-click the Constant icon and enter the new value. Constants contain one outgoing port.

  Note: The Constant is the only editing component that accepts only a single line. This is because the constant will be translated to a Java code string, and the system does not know how to translate multi-line constant input to a Java code string. If multi-line input is required, use the ″\n″ value to separate between lines in the Constant.

  Example: The value ″line1\nline2″ will tell the system to output the text in two lines.

Grouping components: Once you have dragged-and-dropped components onto the canvas to define the desired activity flow, you can select and save the whole or part of this activity flow as a named group. Then later on, you can reuse this

named group in another activity definition just like a regular function block. The following procedure describes the steps to take.

Before you begin: You need to enable "Show child functions in Library window" in the Preference dialog to display the added group.

Perform the following steps:
1. Select one or more graphical components in the canvas.
2. Right-click the canvas to open the Context menu and select Add to My Collection.
3. In the dialog that pops up, enter a name, a description, and select an icon to represent this group.

   Result: The added group will appear in the Library window under My Collection.

Identifying supported function blocks

The supported function blocks are organized into the following categories:
- General/APIs/Business Object Array
- General/APIs/Business Object/Array
- General/APIs/Business Object/Constants
- General/APIs/Business Object
- General/APIs/Database Connection
- General/APIs/Identity Relationship
- General/APIs/Maps/Constants
- General/APIs/Maps/Exception
- General/APIs/Maps
- General/APIs/Participant/Array
- General/APIs/Participant/Constants
- General/APIs/Participant
- General/APIs/Relationship
- General/Date
- General/Date/Formats
- General/Logging and Tracing
- General/Logging and Tracing/Log Error
- General/Logging and Tracing/Log Information
- General/Logging and Tracing/Log Warning
- General/Logging and Tracing/Trace
- General/Mapping
- General/Math
- General/Properties
- General/Relationship
- General/String
- General/Utilities
- General/Utilities/Vector

The following tables identify the function blocks in each category and the acceptable values for their inputs and outputs.

Table 26. General/APIs/Business Object Array

Name	Description	Inputs and outputs with acceptable values
Add Element	Adds a business object to this business object API: BusObjArray.addElement()	Inputs: • business object array--BusObjArray • element--BusObj
Duplicate	Creates a business object array exactly like the original one. API: BusObjArray.duplicate()	Inputs: original--BusObjArray Outputs: duplicate--BusObjArray
Equals	Compares business object array 1's and business object array 2's values, to determine whether they are equal. API: BusObjArray.equals()	Inputs: • array 1--BusObjArray • array 2--BusObjArray Outputs: equal?-- boolean
Get Element At	Retrieves a single business object by specifying its position in the business object array. API: BusObjArray.elementAt()	Inputs: • business object array--BusObjArray • index--int Outputs: element--BusObj
Get Elements	Retrieves the contents of this business object array. API: BusObjArray.getElements()	Inputs: business object array--BusObjArray Outputs: element--BusObj[]
Get Last Index	Retrieves the last available index from a business object array. API: BusObjArray.getLast Index()	Inputs: business object array--BusObjArray Outputs: last index--int
Is Business Object Array	Tests whether value is a business objet array (BusObjArray).	Inputs: value--Object Outputs: result--boolean
Max attribute value	Retrieves the maximum values for the specified attribute among all elements in this business object array. API: BusObjArray.max()	Inputs: • business object array--BusObjArray • attribute--String Outputs: max--String
Min attribute value	Retrieves the minimum value for the specified attribute among all elements in this business object array. API: BusObjArray.min()	Inputs: • business object array--BusObjArray • attribute--String Outputs: min--String
Remove All elements	Removes all elements fro the business object array. API: BusObjArray.removeAllElements()	Inputs: business object array--BusObjArray
Remove Element	Removes a business object element from a business object array. API: BusObjArray.removeElement()	Inputs: • business object array--BusObjArray • element--BusObj
Remove Element At	Removes an element at a particular position in this business object array. API: BusObjArray.removeElementAt()	Inputs: • business object array--BusObjArray • index--int

Table 26. General/APIs/Business Object Array (continued)

Name	Description	Inputs and outputs with acceptable values
Set Element At	Sets the value of a business object in the business object array. API: BusObjArray.setElementAt()	Inputs: • business object array--BusObjArray • index--int • element--BusObj
Size	Gets the number of elements i this business object array. API: BusObjArray.size()	Inputs: business object array--BusObjArray Outputs: size--int
Sum	Adds the values of the specified attribute for all business objects in this business object array. API: BusObjArray.sum()	Inputs: • business object array--BusObjArray • attribute--String Outputs: sum--double
Swap	Reverses the positions of two business objects in this business object array. API: BusObjArray.swap()	Inputs: • business object array--BusObjArray • index 1--int • index 2--int
To String	Retrieves the values in this business object array as a single string. API: BusObjArray.to String()	Inputs: business object array--BusObjArray Outputs: string--String

Table 27. General/APIs/Business Object/Array

Name	Description	Inputs and outputs with acceptable values
Get BusObj At	Retrieves the element at the specified index in the business object array.	Inputs: • array--BusObj[] • index--int Outputs: business object--BusObj
New Business Object Array	Creates a new business object array.	Inputs: size--int Outputs: array--BusObj[]
Set BusObj At	Sets the element at the specified index in the business object array.	Inputs: • array--BusObj[] • index--int • business object--BusObj
Size	Retrieves the size of the business object array	Inputs: array--BusObj[] Outputs: size--int

Table 28. General/APIs/Business Object/Constants

Name	Description	Inputs and outputs with acceptable values
Verb: Create	Business object verb ″Create″.	Outputs: Create--String
Verb: Delete	Business object verb ″Delete″.	Outputs: Delete--String
Verb: Retrieve	Business object verb ″Retrieve″.	Outputs: Retrieve--String
Verb: Update	Business object verb ″Update″.	Outputs: Update--String

Table 29. General/APIs/Business Object

Name	Description	Inputs and outputs with acceptable values
Copy	Copies all attribute values from the input business object. API: BusObj.copy()	Inputs: • copy to--BusObj • copy from--BusObj
Duplicate	Creates a business object exactly like the original one. API: BusObj.duplicate()	Inputs:original--BusObj Outputs: duplicate--BusObj
Equal Keys	Compares business object 1's and business object 2's values, to determine whether they are equal. API: BusObj.equalKeys()	Inputs: • business object 1--BusObj • business object 2--BusObj Outputs: key values equal?-- boolean
Equals	Compares business object 1's and business object 2's values, including child business objects, to determine whether they are equal. API: BusObj.equals()	Inputs: • business object 1--BusObj • business object 2--BusObj Outputs: equal?-- boolean
Exists	Checks for the existence of a business object attribute with a specified name. API: BusObj.exists()	Inputs: • business object--BusObj • attribute--String Outputs: exists?-- boolean
Get Boolean	Retrieves the value of a single attribute, as a boolean, from a business object. API: BusObj.getBoolean()	Inputs: • business object--BusObj • attribute--String Outputs: value-- boolean
Get Business Object	Retrieves the value of a single attribute, as a BusObj, from a business object. API: BusObj.getBusObj()	Inputs: • business object--BusObj • attribute--String Outputs: value--BusObj
Get Business Object Array	Retrieves the value of a single attribute, as a BusObj Array, from a business object. API: BusObj.getBusObjArray()	Inputs: • business object--BusObj • attribute--String Outputs: value--BusObjArray
Get Business Object Type	Retrieves the name of the business object definition on which this business object was based. API: BusObj.getType()	Inputs: business object--BusObj Outputs: type--String
Get Double	Retrieves the value of a single attribute, as a double, from a business object. API: BusObj.getDouble()	Inputs: • business object--BusObj • attribute--String Outputs: value--double

Table 29. General/APIs/Business Object (continued)

Name	Description	Inputs and outputs with acceptable values
Get Float	Retrieves the value of a single attribute, as a float, from a business object. API: BusObj.getFloat()	Inputs: • business object--BusObj • attribute--String Outputs: value--float
Get Int	Retrieves the value of a single attribute, as an integer, from a business object. API: BusObj.getInt()	Inputs: • business object--BusObj • attribute--String Outputs: value--int
Get Long	Retrieves the value of a single attribute, as a long, from a business object. API: BusObj.getLong()	Inputs: • business object--BusObj • attribute--String Outputs: value--long
Get Long Text	Retrieves the value of a single attribute, as a long text, from a business object. API: BusObj.getLongText()	Inputs: • business object--BusObj • attribute--String Outputs: value--String
Get Object	Retrieves the value of a single attribute, as an object, from a business object. API: BusObj.get()	Inputs: • business object--BusObj • attribute--String Outputs: value--Object
Get String	Retrieves the value of a single attribute, as a string, from a business object. API: BusObj.getString()	Inputs: • business object--BusObj • attribute--String Outputs: value--String
Get Verb	Retrieves this business object's verb. API: BusObj.getVerb()	Inputs: business object--BusObj Outputs: verb--String
Is Blank	Finds out whether the value of an attribute is set to a zero-length string. API: BusObj.isBlank()	Inputs: • business object--BusObj • attribute--String Outputs: blank?--boolean
Is Business Object	Tests whether the value is a business object (BusObj).	Inputs: value--Object Outputs: result--boolean
Is Key	Finds out whether a business object's attribute is defined as a key attribute. API: BusObj.isKey()	Inputs: • business object--BusObj • attribute--String Outputs: key?--boolean

Table 29. General/APIs/Business Object (continued)

Name	Description	Inputs and outputs with acceptable values
Is Null	Finds out whether the value of a business object's attribute is null. API: BusObj.isNull()	Inputs: • business object--BusObj • attribute--String Outputs: null?--boolean
Is Required	Finds out whether a business object's attribute is defined as a required attribute.. API: BusObj.isRequired()	Inputs: • business object--BusObj • attribute--String Outputs: required?--boolean
Iterate Children	Iterates through the child business object array.	Inputs: • business object--BusObj • attribute--String • current index--int • current element--BusObj
Key to String	Retrieves the values of a business object's primary key attributes as a string. API: BusObj.keysToString()	Inputs:business object--BusObj Outputs: key string--String
New Business Object	Creates a new business object instance (BusObj) of the specified type. API: Collaboration.BusObj()	Inputs: type--String Outputs: business object--BusObj
Set Content	Sets the contents of this business object to another business object. The two business objects will own the content together. Changes made to one business object will be reflected in the other business object. API: BusObj.setContent()	Inputs: • business object--BusObj • content--BusObj
Set Default Attribute Values	Sets all attributes to their default values. API: BusObj.setDefaultAttrValues()	Inputs:business object--BusObj
Set Keys	Sets the values of the ″to″ business object's key attributes to the values of the key attributes in ″from″ business object. API: BusObj.setKeys()	Inputs: • from business object--BusObj • to business object--BusObj
Set Value with Create	Sets the business object's attribute to a specified value of a particular data type, creating an object for the value if one does not already exist. API: BusObj.setWithCreate()	Inputs: • business object--BusObj • attribute--String • value--BusObj, BusObjArray, Object
Set Verb	Sets the verb of a business object. API: BusObj.setVerb()	Inputs: • business object--BusObj • verb--String
Set Verb with Create	Sets the verb of a child business object, creating the child business object if one does not already exist. API: BusObj.setVerbWithCreate()	Inputs: • business object--BusObj • attribute--String • verb--String

Table 29. General/APIs/Business Object (continued)

Name	Description	Inputs and outputs with acceptable values
Set Value	Sets a business object's attribute to a specified value of a particular data type. API: BusObj.set()	Inputs: • business object--BusObj • attribute--String • value--boolean, double, float, int, long, Object, String, BusObj
Shallow Equals	Compares business object 1 and business object 2's values, excluding child business objects, to determine whether they are equal. API: BusObj.equalsShallow()	Inputs: • business object 1--BusObj • business object 2--BusObj Outputs: equal?--boolean
To String	Gets the values of all attributes in a business object as string. API: BusObj.toString()	Inputs: business object--BusObj Outputs: string--String
Valid Data	Checks whether the specified value is a valid type for a specified attribute. API: BusObj.validData()	Inputs: • business object--BusObj • attribute--String • value--Object, BusObj, BusObjArray, String, long, int, double, float, boolean Outputs: valid?--boolean

Table 30. General/APIs/Database connection

Name	Description	Inputs and outputs with acceptable values
Begin Transaction	Begins an explicit transaction for the current connection. API: CwDBConnection.beginTransaction()	Inputs: database connection--CwDBConnection
Commit	Commits the active transaction associated with the current connection. API: CwDBConnection.commit()	Inputs: database connection--CwDBConnection
Execute Prepared SQL	Executes a prepared SQL Query by specifying its syntax. API: CwDBConnection.executePreparedSQL()	Inputs: • database connection--CwDBConnection • query--String Outputs: equal?-- boolean
Execute Prepared SQL with Parameter	Executes a prepared SQL query by specifying its syntax with the specified parameters. API:CwDBConnection.executePreparedSQL()	Inputs: • database connection--CwDBConnection • query--String • parameters--java.util.Vector
Execute SQL	Executes a static SQL query by specifying its syntax. API: CwDBConnection.executeSQL()	Inputs: • database connection--CwDBConnection • query--String
Execute SQL with Parameter	Executes a static SQL query by specifying its syntax with the specified parameters.. API: CwDBConnection.executeSQL()	Inputs: • database connection--CwDBConnection • query--String • parameters--java.util.Vector

Table 30. General/APIs/Database connection (continued)

Name	Description	Inputs and outputs with acceptable values
Execute Stored Procedure	Executes an SQL stored procedure by specifying its name and parameter array. API: CwDBConnection.executeStored Procedure()	Inputs: • database connection--CwDBConnection • query--String • parameters--java.util.Vector
Get Database Connection	Establishes a connection to a database and returns a CwDBConnection() object. API: BaseDLM.getDBConnection() or BaseCollaboration.getDBConnection()	Inputs: connection pool name--String Outputs: database connection--CwDBConnection
Get Database Connection with Transaction	Establishes a connection to a database and returns a CwDBConnection() object. API: BaseDLM.getDBConnection() or BaseCollaboration.getDBConnection()	Inputs: • connection pool name--String • implicit transaction--boolean Outputs:database connection--CwDBConnection
Get Next Row	Gets the next row from the query result. API: CwDBConnection.nextRow()	Inputs: database connection--CwDBConnection Outputs: row--java.util.Vector
Get Update Count	Gets the number of rows affected by the last write operation to the database. API: CwDBConnection.getUpdateCount()	Inputs: database connection--CwDBConnection Outputs: count--int
Has More Rows	Determines whether the query result has more rows to process. API: CwDBConnection.hasMoreRows()	Inputs: database connection--CwDBConnection Outputs: more rows?--boolean
In Transaction	Determines whether a transaction is in progress in the current connection. API: CwDBConnection.inTransaction()	Inputs: database connection--CwDBConnection Outputs: in transaction?--boolean
Is Active	Determines whether the current connectio is active. API: CwDBConnection.isActive()	Inputs: database connection--CwDBConnection Outputs: is active?--boolean
Release	Releases use of the current connection, returning it to its connection pool. API: CwDBConnection.release()	Inputs: database connection--CwDBConnection
Roll Back	Rolls back the active transaction associated with the current connection. API: CwDBConnection.rollback()	Inputs: database connection--CwDBConnection

Table 31. General/APIs/Identity Relationship

Name	Description	Inputs and outputs with acceptable values
Add My Children	Adds the specified child instances to a parent/child relationship for an identity relationship. API: IdentityRelationship.addMyChildren()	Inputs: • map--BaseDLM • parentChildRelDefName--String • parentParticipantDefName--String • parentBusObj--BusObj • childParticipantDefName--String • childBusObjList--BusObj,BusObjArray
Delete All My Chidren	Removes all child instances to a parent/child relationship for an identity relationship belonging to the specified parent. API: IdentityRelationship.deleteMyChildren()	Inputs: • map--BaseDLM • parentChildRelDefName--String • parentParticipantDefName--String • parentBusObj--BusObj • childParticipantDefName--String
Delete My Children	Removes the specified child instances to a parent/child relationship for an identity relationship belonging to the specified parent. API: IdentityRelationship.deleteMyChildren()	Inputs: • map--BaseDLM • parentChildRelDefName--String • parentParticipantDefName--String • parentBusObj--BusObj • childParticipantDefName--String • childBusObjList--BusObj,BusObjArray
Foreign Key Cross-Reference	Performs a lookup in the relationship table in the relationship database based on the foreign key of the source business object, adding a new relationship instance in the foreign relationship table if the foreign key does not exist. API: IdentityRelationship.foreignKeyXref()	Inputs: • map--BaseDLM • RelDefName--String • appParticipantDefName--String • genParticipantDefName--String • appSpecificBusObj--BusObj • appForeignAttr--String • genericBusObj--BusObj • genForeignAttr--String
Foreign Key Lookup	Performs a lookup in a foreign relationship table based on the foreign key of the source business object, failing to find a relationship instance if the foreign key does not exist in the foreign relationship table.. API: IdentityRelationship.foreignKeyLookup()	Inputs: • map--BaseDLM • relDefName--String • appParticipantDefName--String • appSpecificBusObj--BusObj • appForeignAttr--String • genericBusObj--BusObj • genForeignAttr--String

Table 31. General/APIs/Identity Relationship (continued)

Name	Description	Inputs and outputs with acceptable values
Maintain child Verb	Sets the child business object verb based on the map execution context and the verb of the parent business object. API: IdentityRelationship.maintainChildVerb()	Inputs: • map--BaseDLM • relDefName--String • appSpecificParticipantName--String • genericParticipantName--String • appSpecificObj--BusObj • appSpecificChildObj--String • genericObj--BusObj • genericChildObj--String • to_Retrieve--boolean • Is_Composite--boolean
Maintain Composite Relationship	Maintains a composite identity relationship from within the parent map. API: IdentityRelationship.maintain CompositeRelationship()	Inputs: • map--BaseDLM • relDefName--String • participantDefName--String • appSpecificBusObj--BusObj • genericBusObjList--BusObj, BusObjArray
Maintain Simple Identity Relationship	Maintains a simple identity relationship from within either a parent or child map. API: IdentityRelationship.maintain Simple Identity Relationship()	Inputs: • map--BaseDLM • relDefName--String • participantDefName--String • appSpecificBusObj--BusObj • genericBusObj--BusObj
Update My Children	Adds and deletes child instances in a specified parent/child relationship of an identity relationship as necessary. API: IdentityRelationship.updateMyChildren()	Inputs: • map--BaseDLM • parentChildRelDefName--String • parentParticipantDef--String • parentBusObj--BusObj • childParticipantDef--String • childAttrName--String • childIdentityRelDefName--String • childIdentityParticipantDefName--String

Table 32. General/APIs/Maps/Constants

Name	Description	Inputs and outputs with acceptable values
Calling Context: ACCESS _REQUEST	An access client has sent an access request from an external application to InterChange Server Express. API: MapExeContext.ACCESS_REQUEST	Outputs: ACCESS_REQUEST--String
Calling Context: ACCESS _RESPONSE	The source business object is sent back to the source access client in response to a subscription delivery request. API: MapExeContext.ACCESS_RESPONSE	Outputs: ACCESS_RESPONSE--String

Table 32. General/APIs/Maps/Constants (continued)

Name	Description	Inputs and outputs with acceptable values
Calling Context: EVENT _DELIVERY	A connector has sent an event from the application to InterChange Server Express (event-triggered flow). API: MapExeContext.EVENT_DELIVERY	Outputs: EVENT_DELIVERY--String
Calling Context: SERVICE_CALL _FAILURE	A collaboration's service call request has failed. As such, corrective action might need to be performed. API: MapExeContext.SERVICE_CALL_FAILURE	Outputs: SERVICE_CALL_FAILURE --String
Calling Context: SERVICE_CALL _REQUEST	A collaboration is sending a business object down to the application through a service call request. API: MapExeContext.SERVICE_CALL_REQUEST	Outputs: SERVICE_CALL_REQUEST --String
Calling Context: SERVICE_CALL _RESPONSE	A business object was received from the application as a result of a successful response to a collaboration service call request. API: MapExeContext.SERVICE_CALL _RESPONSE	Outputs: SERVICE_CALL_RESPONSE --String

Table 33. General/APIs/Maps/Exception

Name	Description	Inputs and outputs with acceptable values
Raise Map Exception	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception type--String • message--String
Raise Map Exception 1	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception type--String • message--String • parameter 1--String
Raise Map Exception 2	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception type--String • message--String • parameter 1--String • parameter 2--String
Raise Map Exception 3	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception type--String • message--String • parameter 1--String • parameter 2--String • parameter 3--String

Table 33. General/APIs/Maps/Exception (continued)

Name	Description	Inputs and outputs with acceptable values
Raise Map Exception 4	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception type--String • message--String • parameter 1--String • parameter 2--String • parameter 3--String • parameter 4--String
Raise Map Exception 5	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception type--String • message--String • parameter 1--String • parameter 2--String • parameter 3--String • parameter 4--String • parameter 5--String
Raise Map RunTimeEntity Exception	Raises a map run-time exception. API: raiseException()	Inputs: • map--BaseDLM • exception--RunTimeEntityException

Table 34. General/APIs/Maps

Name	Description	Inputs and outputs with acceptable values
Get Adapter Name	Retrieves the adapter name associated with the current map instance.. API: MapExeContext.getConnName()	Inputs: map--BaseDLM Outputs: adapter name--String
Get Calling Context	Retrieves the calling context associated with the current map instance.. API: MapExeContext.getInitiator()	Inputs: map--BaseDLM Outputs: calling context--String
Get Original Request Business Object	Retrieves the original-request business object associated with the current map instance.. API: MapExeContext.getOriginalRequestBO()	Inputs: map--BaseDLM Outputs: original business object--BusObj

Table 35. General/APIs/Participant/Array

Name	Description	Inputs and outputs with acceptable values
Get Participant At	Retrieves the element at the specified index in the participant array.	Inputs: • array--Server.RelationshipServices.Participant[] • index--int Outputs: participant--Server.RelationshipServices.Participant

Table 35. General/APIs/Participant/Array (continued)

Name	Description	Inputs and outputs with acceptable values
New Participant Array	Creates a new participant array with the specified size.	Inputs: size--int Outputs: array--Server.RelationshipServices.Participant[]
Set Participant At	Sets the element at the specified index in the participant array.	Inputs: • array--Server.RelationshipServices.Participant[] • index--int • participant--Server.RelationshipServices.Participant
Size	Retrieves the size of the participant array.	Inputs: array--Server.RelationshipServices.Participant[] Outputs: size--int

Table 36. General/APIs/Participant/Constants

Name	Description	Inputs and outputs with acceptable values
Participant: INVALID _INSTANCE_ID	Participant constant indicating the participant ID is invalid.. API: Participant.INVALID_INSTANCE_ID	Outputs: INVALID_INSTANCE_ID--int

Table 37. General/APIs/Participant

Name	Description	Inputs and outputs with acceptable values
Get Boolean Data	Retrieves the data associated with the Participant object. API: Participant.getBoolean()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--boolean
Get Business Object Data	Retrieves the data associated with the Participant object. API: Participant.getBusObj()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--BusObj
Get Double Data	Retrieves the data associated with the Participant object. API: Participant.getDouble()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--double
Get Float Data	Retrieves the data associated with the Participant object. API: Participant.getFloat()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--float
Get Instance ID	Retrieves the relationship instance ID of the relationship in which the participant instance is participating. API: Participant.getInstanceId()	Inputs: participant--Server.RelationshipServices.Participant Outputs: instance ID--int
Get Int Data	Retrieves the data associated with the Participant object. API: Participant.getInt()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--int

Table 37. General/APIs/Participant (continued)

Name	Description	Inputs and outputs with acceptable values
Get Long Data	Retrieves the data associated with the Participant object. API: Participant.getLong()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--long
Get Participant Name	Retrieves the participant definition name from which the participant instance is created. API: Participant.getParticipantDefinition()	Inputs: participant--Server.RelationshipServices.Participant Outputs: name--String
Get Relationship Name	Retrieves the name of the relationship definition in which the participant instance is participating. API: Participant.getRelationshipDefinition()	Inputs: participant--Server.RelationshipServices.Participant Outputs: name--String
Get String Data	Retrieves the data associated with the Participant object. API: Participant.getString()	Inputs: participant--Server.RelationshipServices.Participant Outputs: data--String
New Participant	Creates a new participant instance with no relationship instance. API: Participant()	Inputs: • relDefName--String • partDefName--String • partData--BusObj, String, long, int, double, float, boolean Output: participant--Server.RelationshipServices.Participant
New Participant in Relationship	Creates a new participant instance for adding to an existing participant in a relationship instance. API: RelationshipServices.Participant()	Inputs: • relDefName--String • partDefName--String • instanceId--int • partData--BusObj, String, long, int, double, float, boolean Output: participant--Server.RelationshipServices.Participant
Set Data	Sets the data associated with the participant instance. API: Participant.set()	Inputs: • participant--Server.RelationshipServices.Participant • partData--BusObj, String, long, int, double, float, boolean
Set Instance ID	Sets the instance ID of the relationship in which the participant instance is participating. API: Participant.setInstanceId()	Inputs: • participant--Server.RelationshipServices.Participant • id--int
Set Participant Defnition	Sets the participant definition name from which the participant instance is created. API: Participant.setParticipantDefinition()	Inputs: • participant--Server.RelationshipServices.Participant • partDefName--String
Set Relationship Definition	Sets the relationship definition in which the participant instance is participating. API: Participant.setRelationshipDefinition()	Inputs: • participant--Server.RelationshipServices.Participant • relDefName--String

Table 38. General/APIs/Relationship

Name	Description	Inputs and outputs with acceptable values
Add Participant	Adds an existing participant object to a relationship instance. API: Relationship.addParticipant()	Inputs: participant--Server.RelationshipServices.Participant Outputs: result instance ID--int
Add Participant Data	Adds a new participant to an existing relationship instance. API: Relationship.addParticipant()	Inputs: • relDefName--String • partDefName--String • instanceId--int • partData--BusObj, String, long, int, double, float, boolean Outputs: result instance ID--int
Add Participant Data to new Relationship	Adds a participant to a new relationship instance. API: Relationship.addParticipant()	Inputs: • relDefName--String • partDefName--String • partData--BusObj, String, long, int, double, float, boolean Outputs: result instance ID--int
Create Relationship	Creates a new relationship instance. API: Relationship.create()	Inputs: • relDefName--String • partDefName--String • partData--BusObj, String, long, int, double, float, boolean Outputs: instance ID--int
Create Relationship with Participant	Creates a new relationship instance. API: Relationship.create()	Inputs: participant--Server.RelationshipServices.Participant Outputs: instance ID--int
Deactivate Participant	Deactivates a participant from one or more relationship instances. API: Relationship.deactivate Participant()	Inputs: participant--Server.RelationshipServices.Participant
Deactivate Participant By Data	Deactivates a participant from one or more relationship instances. API: Relationship.deactivate Participant()	Inputs: • relDefName--String • partDefName--String • partData--BusObj, String, long, int, double, float, boolean
Deactivate Participant by Instance	Deactivates a participant from a specific relationship instance. API: Relationship.deactivate ParticipantByInstance()	Inputs: • relDefName--String • partDefName--String • instanceId--int

Table 38. General/APIs/Relationship (continued)

Name	Description	Inputs and outputs with acceptable values
Deactivate Participant By Instance Data	Deactivates a participant from a specific relationship instance with the data associated with the participant. API: Relationship.deactivate ParticipantByInstance()	Inputs: • relDefName--String • partDefName--String • instanceId--int • partData--BusObj, String, long, int, double, float, boolean
Delete Participant	Removes a participant instance from one or more relationship instances. API: Relationship.deleteParticipant()	Inputs: participant--Server.RelationshipServices.Participant
Delete Participant By Instance	Removes a participant from a specific relationship instance. API: Relationship.deleteParticipanByInstancet()	Inputs: • relDefName--String • partDefName--String • instanceId--int
Delete Participant By Instance Data	Removes a participant from a specific relationship instance with the dta associated with the participant. API: Relationship.deleteParticipanByInstancet()	Inputs: • relDefName--String • partDefName--String • instanceId--int • partData--BusObj, String, long, int, double, float, boolean
Delete Participant with Data	Removes a participant instance from one or more relationship instances. API: Relationship.deleteParticipant()	Inputs: • relDefName--String • partDefName--String • partData--BusObj, String, long, int, double, float, boolean
Get Next Instance ID	Returns the next available relationship instance ID for a relationship, based on the relationship definition name. API: Relationship.getNewID()	Inputs: relDefName--String Outputs: ID--int
Retrieve Instances	Retrieves zero or more IDs of relationship instances which contain the given participant(s). API: Relationship.retrieveInstances()	Inputs: • relDefName--String • partDefName--String,String[] • partData--BusObj, String, long, int, double, float, boolean Outputs: instance IDs--int
Retrieve Instances for Participant	Retrieves zero or more IDs of relationship instances which contain a given participant. API: Relationship.retrieveInstances()	Inputs: • relDefName--String • partData--BusObj, String, long, int, double, float, boolean Outputs: instance IDs--int

Table 38. General/APIs/Relationship (continued)

Name	Description	Inputs and outputs with acceptable values
Retrieve Participants	Retrieves zero or more participants from a relationship instance. API: Relationship.retrieveParticipants()	Inputs: • relDefName--String • partDefName--String, String[] • instanceId--int Outputs: participant instances--Server.RelationshipServices.Participant[]
Retrieve Participants with ID	Retrieves zero or more participants from a relationship instance. API: Relationship.retrieveParticipants()	Inputs: • relDefName--String • instanceId--int Outputs: participant instances--Server.RelationshipServices.Participant[]
Update Participant	Updates a participant in one or more relationhip instances. API: Relationship.updateParticipant()	Inputs: • relDefName--String • partDefName--String • partData--BusObj
Update Participant By Instance	Updates a participant in a specific relationship instance. API: Relationship.updateParticipantByInstance()	Inputs: • relDefName--String • partDefName--String, String[] • instanceId--int
Update Participant By Instance Data	Updates a participant in a specific relationship instance with the data associated with the participant. API: Relationship.updateParticipantByInstance()	Inputs: • relDefName--String • partDefName--String • instanceId--int • partData--BusObj, String

Table 39. General/Date

Name	Description	Inputs and outputs with acceptable values
Add Day	Adds additional days to the from date.	Inputs: • from date--String • date format--String • day to add--int Outputs: to date-- String
Add Month	Adds additional months to the from date.	Inputs: • from date--String • date format--String • month to add--int Outputs: to date-- String
Add Year	Adds additional years to the from date.	Inputs: • from date--String • date format--String • year to add--int Outputs: to date-- String

Table 39. General/Date (continued)

Name	Description	Inputs and outputs with acceptable values
Date After	Compares two dates and determines whether Date 1 is after Date 2.	Inputs: • Date 1--String • Date 1 format--String • Date 2--String • Date 2 format--String Outputs: Is Date 1 after Date 2?-- boolean
Date Before	Compares two dates and determines whether Date 1 is before Date 2.	Inputs: • Date 1--String • Date 1 format--String • Date 2--String • Date 2 format--String Outputs: Is Date 1 before Date 2?-- boolean
Date Equals	Compares two dates and determines whether they are equal.	Inputs: • Date 1--String • Date 1 format--String • Date 2--String • Date 2 format--String Outputs: Are they equal?-- boolean
Format Change	Changes a date format.	Inputs: • date--String • input format--String • output format--String Outputs: formatted date--String
Get Day	Returns the numeric day of month based on date expression.	Inputs: • Date--String • Format--String Outputs: Day--int
Get Month	Returns the numeric month of year based on date expression.	Inputs: • Date--String • Format--String Outputs: Month--int
Get Year	Returns the numeric year based on date expression.	Inputs: • Date--String • Format--String Outputs: Year--int

Table 39. General/Date (continued)

Name	Description	Inputs and outputs with acceptable values
Get Year Month Day	Given an input date, extracts the Year/Month/Day parts from the input date respectively.	Inputs: • Date--String • Format--String Outputs: • Year--int • Month--int • Day--int
Now	Gets today's date.	Inputs: format--String Outputs: now--String

Table 40. General/Date/Formats

Name	Description	Inputs and outputs with acceptable values
yyyy-MM-dd	Date format of yyyy-MM-dd **Example:** 2003-02-25	Outputs: format--String
yyyyMMdd	Date format of yyyyMMdd **Example:** 20030225	Outputs: format--String
yyyyMMdd HH:mm:ss	Date format of yyyyMMdd HH:mm:ss **Example:** 20030225 12:36:40	Outputs: format--String

Table 41. General/Logging and Tracing

Name	Description	Inputs and outputs with acceptable values
Log error	Sends the specified error message to the ICS log file.	Inputs: message--String, byte, short, int, long, float, double
Log error ID	Sends the error message associated with the specified ID to the ICS log file.	Inputs: ID--String, byte, short, int, long, float, double
Log information	Sends the specified information message to the ICS log file.	Inputs: message--String, byte, short, int, long, float, double
Log information ID	Sends the information message associated with the specified ID to the ICS log file.	Inputs: ID--String, byte, short, int, long, float, double
Log warning	Sends the specified warning message to the ICS log file	Inputs: message--String, byte, short, int, long, float, double
Log warning ID	Sends the warning message associated with the specified ID to the ICS log file.	Inputs: ID--String, byte, short, int, long, float, double
Trace	Sends the specified trace message to the ICS log file.	Inputs: message--String, byte, short, int, long, float, double

Table 42. General/Logging and Tracing/Log Error

Name	Description	Inputs and outputs with acceptable values
Log error ID 1	Formats the error message associated with the specified ID with the parameter and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter--String, byte, short, int, long, float, double

Table 42. General/Logging and Tracing/Log Error (continued)

Name	Description	Inputs and outputs with acceptable values
Log error ID 2	Formats the error message associated with the specified ID with the parameters and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double
Log error ID 3	Formats the error message associated with the specified ID with the parameters and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double • parameter 3--String, byte, short, int, long, float, double

Table 43. General/Logging and Tracing/Log Information

Name	Description	Inputs and outputs with acceptable values
Log information ID 1	Formats the information message associated with the specified ID with the parameter and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter--String, byte, short, int, long, float, double
Log information ID 2	Formats the information message associated with the specified ID with the parameters and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double
Log information ID 3	Formats the information message associated with the specified ID with the parameters and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double • parameter 3--String, byte, short, int, long, float, double

Table 44. General/Logging and Tracing/Log Warning

Name	Description	Inputs and outputs with acceptable values
Log warning ID 1	Formats the warning message associated with the specified ID with the parameter and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter--String, byte, short, int, long, float, double

Table 44. General/Logging and Tracing/Log Warning (continued)

Name	Description	Inputs and outputs with acceptable values
Log warning ID 2	Formats the warning message associated with the specified ID with the parameters and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double
Log warning ID 3	Formats the warning message associated with the specified ID with the parameters and send it to the ICS log file.	Inputs: • ID--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double • parameter 3--String, byte, short, int, long, float, double

Table 45. General/Logging and Tracing/Trace

Name	Description	Inputs and outputs with acceptable values
Trace ID 1	Formats the trace message associated with the specified ID with the parameter and display it if tracing is set to the specified level or a higher level.	Inputs: • ID--String, byte, short, int, long, float, double • level--String, byte, short, int, long, float, double • parameter--String, byte, short, int, long, float, double
Trace ID 2	Formats the trace message associated with the specified ID with the parameters and display it if tracing is set to the specified level or a highter level.	Inputs: • ID--String, byte, short, int, long, float, double • level--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double
Trace ID 3	Formats the trace message associated with the specified ID with the parameters and display it if tracing is set to the specified level or a highter level.	Inputs: • ID--String, byte, short, int, long, float, double • level--String, byte, short, int, long, float, double • parameter 1--String, byte, short, int, long, float, double • parameter 2--String, byte, short, int, long, float, double • parameter 3--String, byte, short, int, long, float, double

Table 45. General/Logging and Tracing/Trace (continued)

Name	Description	Inputs and outputs with acceptable values
Trace on Level	Displays the trace message if tracing is set to the specified level or a higher level.	Inputs: • message--String, byte, short, int, long, float, double • level--String, byte, short, int, long, float, double

Table 46. General/Mapping

Name	Description	Inputs and outputs with acceptable values
Run Map	Executes the specified map with the current calling context.	Inputs: • Map Name--String • Source Business Objects--BusObj, BusObj[] Outputs: Map Results--BusObj, BusObj[]
Run Map with Context	Executes the specified map with the calling context specified.	Inputs: • Map Name--String • Source Business Objects--BusObj, BusObj[] • calling context--String Outputs: Map Results--BusObj, BusObj[]

Table 47. General/Math

Name	Description	Inputs and outputs with acceptable values
Absolute value	a=abs(b) API: Math.abs()	Inputs: b--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Ceiling	Returns the next highest integer that is greater than or equal to the specified numeric expression.	Inputs: number--String, float, double Outputs: ceiling--int
Divide	a=b/c	Inputs: • b--byte, short, int, long, float, double • c--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Equal	Is value 1 equal to value 2?	Inputs: • value 1--String, byte, short, int, long, float, double • value 2--String, byte, short, int, long, float, double Outputs: are they equal?--boolean
Floor	Returns the next lowest integer that is greater than or equal to the specified numeric expression.	Inputs: number--String, float, double Outputs: floor--int

Table 47. General/Math (continued)

Name	Description	Inputs and outputs with acceptable values
Greater than	Is value 1 greater than value 2?	Inputs: • value 1--byte, short, int, long, float, double • value 2--byte, short, int, long, float, double Outputs: result--boolean
Greater than or Equal	Is value 1 greater than or equal to value 2?	Inputs: • value 1--byte, short, int, long, float, double • value 2--byte, short, int, long, float, double Outputs: result--boolean
Less than	result=value 1 is less than value 2?	Inputs: • value 1--byte, short, int, long, float, double • value 2--byte, short, int, long, float, double Outputs: result--boolean
Less than or equal	Is value 1 less than or equal to value 2?	Inputs: • value 1--byte, short, int, long, float, double • value 2--byte, short, int, long, float, double Outputs: result--boolean
Maximum	a=max(b, c) API: Math.max()	Inputs: • b--byte, short, int, long, float, double • c--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Minimum	a=min(b, c) API: Math.min()	Inputs: • b--byte, short, int, long, float, double • c--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Minus	a=b-c	Inputs: • b--byte, short, int, long, float, double • c--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Multiply	a=b*c	Inputs: • b--byte, short, int, long, float, double • c--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Not Equal	result=is value 1 not equal to value 2?	Inputs: • value 1--String, byte, short, int, long, float, double • value 2--String, byte, short, int, long, float, double Outputs: are they not equal?--boolean

Table 47. General/Math (continued)

Name	Description	Inputs and outputs with acceptable values
Not a Number	Returns true if input is not a number.	Inputs: input--String Outputs: is not a number--boolean
Number to String	Converts a numeric expression to a character expression.	Inputs: number--String, short, int, long, float, double Outputs: string--String
Plus	a=b+c	Inputs: • b--byte, short, int, long, float, double • c--byte, short, int, long, float, double Outputs: a--byte, short, int, long, float, double
Round	Rounds a numeric expression down to the next lowest integer if <5; otherwise, the integer is rounded up.	Inputs: number--String, float, double Outputs: rounded number--int
String to Number	Converts a character expression to a numeric expression. API: Math.type()	Inputs: string--String Outputs: String, short, int, long, float, double

Table 48. General/Properties

Name	Description	Inputs and outputs with acceptable values
Get Property	Retrieves the specified configuration property value.	Inputs: property name--String Outputs: property value--String

Table 49. General/Relationship

Name	Description	Inputs and outputs with acceptable values
Maintain Identity Relationship	Maintain Identity Relationship with the maintainSimpleIdentityRelationship() Relationship API.	Inputs: • relationship name--String • participant name--String • Generic Business Object--String • Application-Specific Business Object--String • calling context--String
Static Lookup	Look up a static value in the relationship.	Inputs: • relationship name--String • participant name--String • inbound?--boolean • source value--String Outputs: lookup value--String

Table 50. General/String

Name	Description	Inputs and outputs with acceptable values
Append Text	Appends the ″in string2″ to the end of the string ″in string 1.″	Inputs: • in string 1--String • in string 2--String Outputs: result--String
If	Returns the first value if condition is true and the second value if condition is false.	Inputs: • condition--boolean, Boolean • value 1--String • value 2--String Outputs: result--String
Is Empty	Returns the second value if the first value is empty.	Inputs: • value 1--String • value 2--String Outputs: result--String
Is NULL	Returns the second value if the first value is null.	Inputs: • value 1--String • value 2--String Outputs: result--String
Left Fill	Returns a string of the specifed length; fills the left with indicated value.	Inputs: • string--String • fill string--String • length--int Outputs: filled string--String
Left String	Returns the left portion of string for the specified number of positions.	Inputs: • string--String • length--int Outputs: left string--String
Lower Case	Changes all characters to Lower Case letters	Inputs: fromString--String Outputs: toString--String
Object To String	Gets a string representation of the object.	Inputs: object--Object Outputs: string--String
Repeat	Returns a character string that contains a specified character expression repeated a specified number of times.	Inputs: • repeating string--String • repeat count--int Outputs: result--String
Replace	Replaces part of a string with indicated value data.	Inputs: • input--String • old string--String • new string--String Outputs: replaced string--String

Table 50. General/String (continued)

Name	Description	Inputs and outputs with acceptable values
Right Fill	Returns a string of the specified length; fills the right with indicated value.	Inputs: • string--String • fill string--String • length--int Outputs: filled string--String
Right String	Returns the right portion of string for the specified number of positions.	Inputs: • string--String • length--int Outputs: right string--String
Substring by position	Returns a portion of the string based on start and end parameters.	Inputs: • string--String • start positon--int • end position--int Outputs: substring--String
Substring by value	Returns a portion of the string based on start and end parameters. The substring will not include the start and end value.	Inputs: • string--String • start value--int • end value--int Outputs: substring--String
Text Equal	Compares the strings ″inString1″ and ″inString2″ and determine whether they are the same.	Inputs: • inString1--String • inString2--String Outputs: are they equal?--boolean
Text Equal Ignore Case	Compares the strings ″inString1″ and ″inString2″ lexicographically, ignoring case considerations.	Inputs: • inString1--String • inString2--String Outputs: are they equal?--boolean
Text Length	Finds the total number of characters in a String	Inputs: str--String Outputs: length---byte, short, int, long, float, double
Trim Left	Trims the specified number of characters from the left side of the string.	Inputs: • input--String • trim length--int Outputs: result--String
Trim Right	Trims the specified number of characters from the right side of the string.	Inputs: • input--String • trim length--int Outputs: result--String
Trim Text	Trims white spaces before and after string	Inputs: in string--String Outputs: trimmed string--String

Table 50. General/String (continued)

Name	Description	Inputs and outputs with acceptable values
Upper Case	Changes all characters into Upper Case letters	Inputs: fromString--String Outputs: toString--String

Table 51. General/Utilities

Name	Description	Inputs and outputs with acceptable values
Catch Error	Catches all the Exceptions thrown in the current activity and its subactivities. (Double-click the function block icon in the canvas to define the subactivity.)	Inputs: • Error Name--String • Error Message--String
Catch Error Type	Catches the specified Exception type thrown in the current activity and its subactivities. (Double-click the function block icon in the canvas to define the subactivity.)	Inputs: • error type--String • Error Message--String
Condition	If ″Condition″ is true, executes the subactivity defined in″True Action″; otherwise, executes the subactivity defined in ″False Action.″ (Double-click the function block icon in the canvas to define the subactivity.)	Inputs: Condition--boolean
Loop	Repeats the subactivity until ″Condition″ is false. (Double-click the function block icon in the canvas to define the subactivity.)	Inputs: Condition--boolean
Move Attribute in Child	Moves the value from ″from attribute″ to ″to attribute″.	Inputs: • source parent--BusObj • source child BO attribute--string • from attribute--String • destination parent--BusObj • destination child BO attribute--String • to attribute--String
Raise Error	Throws a new Java Exception with the given message.	Inputs: message--String
Raise Error Type	Throws the specified Java Exception with the given message.	Inputs: • error type--String • message--String

Table 52. General/Utilities/Vector

Name	Description	Inputs and outputs with acceptable values
Add Element	Adds the specified element to the end of the vector, increasing its size by one.	Inputs: vector--java.util.Vector Outputs: element--Object
Get Element	Gets the element at the specified index in the Vector object.	Inputs: • vector--java.util.Vector • index--int Outputs: element--Object

Table 52. General/Utilities/Vector (continued)

Name	Description	Inputs and outputs with acceptable values
Iterate Vector	Iterates through the vector object.	Inputs: • vector--java.util.Vector • current index--int • current element--Object
New Vector	Creates a new Vector object.	Outputs: vector--java.util.Vector
Size	Gets the number of elements in this vector.	Inputs: vector--java.util.Vector Outputs: size--int
To Array	Gets the array representation containing all of the elements in this vector.	Inputs: vector--java.util.Vector Outputs: array--Object[]

Example 1 of using the Activity Editor

The following example illustrates the steps for using Activity Editor to change the source attribute's value to all uppercase and assign the change to the destination attribute.

Perform the following steps:

1. From the Diagram tab, drag the source attribute onto the destination attribute to create a Custom transformation rule. Then click the icon of the Custom transformation rule to open Activity Editor.

   Example: Figure 47 shows the Custom transformation we are using in this example. The source attribute is ObjClarify_contact.LastName, and the destination attribute is ObjContact.LastName.

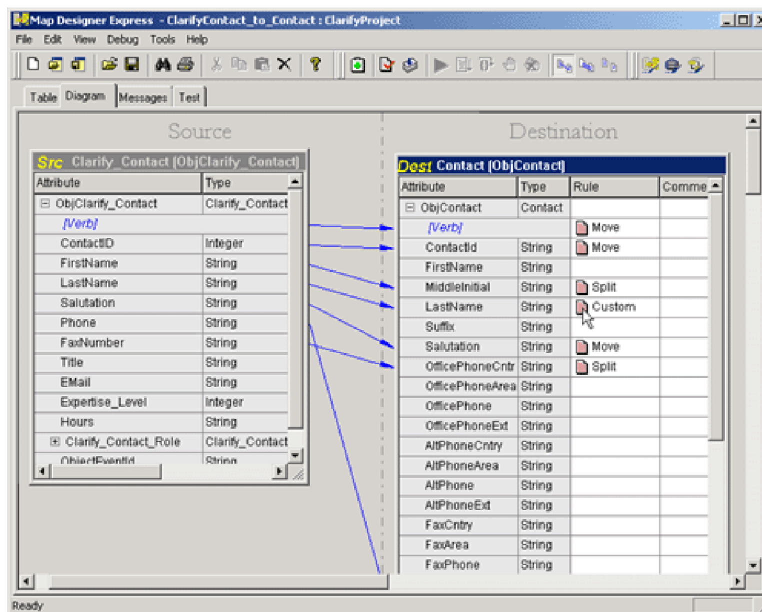


Figure 47. Custom transformation rule

   Result: Activity Editor opens.

   For more information on creating Custom and other transformations, see Chapter 2, "Creating maps," on page 13..

2. Select a category in the Library window (top left) to show the available function blocks in that category in the Content window (bottom left).

   Figure 48 shows the available functions blocks for the ″String″ category; the source and destination attributes in our example are displayed as icons in the graphical canvas.

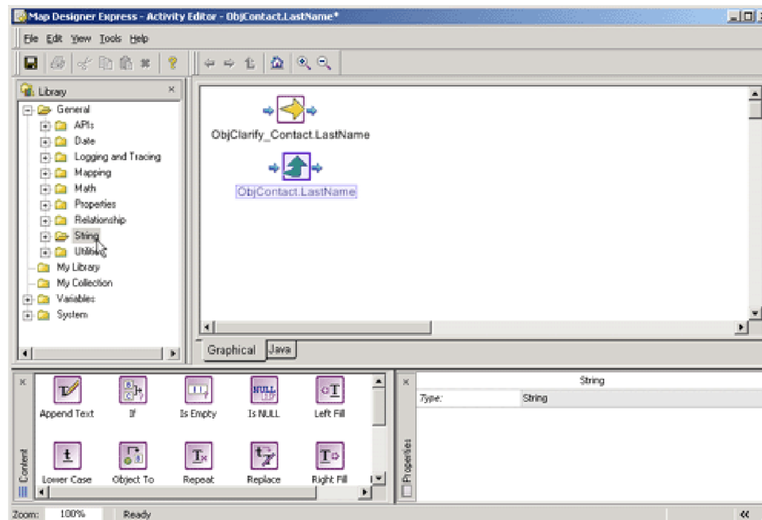


Figure 48. Function blocks in String category and icons for source and destination attributes

3. To use any of the function blocks in the activity, drag the function block from the tree in the Library window and drop it onto the activity canvas; or alternatively, drag the icon from the Content window and drop it onto the activity canvas.

   Example: In our example, we want to change the source attribute to all uppercase letters, so we will drag-and-drop the Upper Case function block in the String category from the Content window onto the activity canvas, as shown in Figure 49..

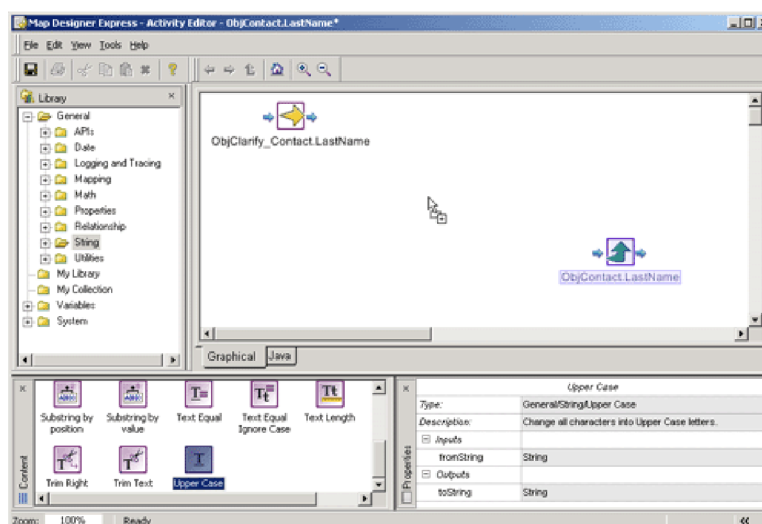


Figure 49. Dragging the Upper Case function block

4. After you drop a function block on the activity canvas, you can move it around the canvas by selecting the function block icon and dragging and dropping it at the desired position. When the function block is in place, you are ready to connect the inputs and outputs of the function block to define the flow of execution.

   Example: In our example, we want to convert the attribute value of ObjClarify_Contact.LastName to all uppercase letters. We can do this by connecting the output of the icon for ObjClarify_Contact.LastName to the input of the Upper Case function block. To do this, move the mouse cursor to the output of the icon of port ObjClarify_Contact.LastName.

   Result: The shape of the icon will change to an arrow to indicate that you can initiate a link at that point, as shown in Figure 50..

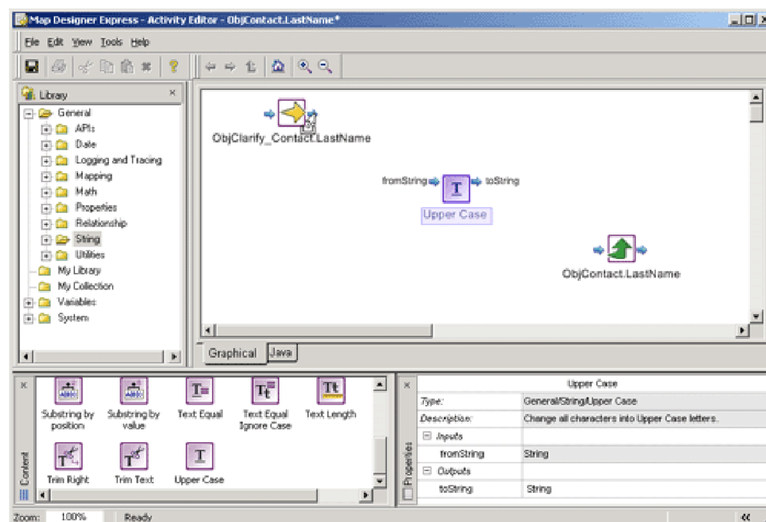


Figure 50. Cursor as arrow at output port of ObjClarify_Contact.LastName

5. When the mouse icon is changed to an arrow, hold down the mouse button and move the mouse to the input of the Upper Case function block, and release the mouse button. A connection link will be drawn to connect the input and outputs.

   To indicate that the result of the Upper Case function block should be assigned to the destination attribute (in our example, ObjContact.LastName), repeat the same steps to drag-and-drop from the output of the Upper Case function block to the input of the ObjContact.LastName port icon. Figure 51 shows the

connection links.

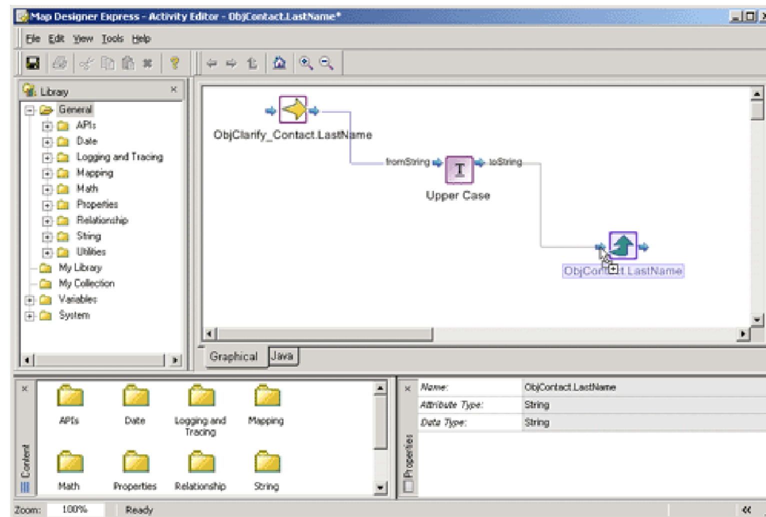


Figure 51. Upper Case function block with connection links

Result: We have defined an activity which will take the value of the source attribute, uppercase it, and set the upper-cased value to the destination attribute.

6. Save the activity by selecting To Project or To File from the File-->Save submenu or by clicking the Save Map to Project or Save Map to File button in the Standard toolbar.
7. To see an example of the Java code that will be generated by this activity, click the Java tab.

   Result: The Java tab will be activated with the sample Java code, as shown in Figure 52..

Map Designer Express - Activity Editor - ObjContact.LastName

File Edit View Tools Help

```
{
    String var_2 = null;
    String var_5 = null;
    String var_9 = null;
    {
        var_2 = ObjClarify_Contact.get("LastName") == null ? "" : ObjClarify_Contact.get("LastName")
    }
    var_5 = var_2.toUpperCase();
    var_9 = var_5;
    {
        {
            BusObj destBusObj = ObjContact;
            String destAttr = "LastName";
            //
            // Set the destination value only if neither
            // source nor destination are null.
            //
            if ((var_9 != null) && (destBusObj != null)) {
                if (dataValidationLevel >= 1) {
                    if (!ObjContact.validData("LastName", var_9)) {
                        //
                        // Log a warning about this failure.
                        //
                        String warningMessage = "Invalid data encountered when attempting to set
                        //
                        // Log a warning about this failure.
                        //
```

Graphical Java

Zoom: 100% Ready 1 : 1

Figure 52. Java tab with code

Example 2 of using the Activity Editor

The following example illustrates the steps for using Activity Editor to change the source value's date format to a different format and assign it to the destination attribute.

Perform the following steps:

1. From the Diagram tab, drag the source attribute onto the destination attribute to create a Custom transformation rule. Then click the icon of the Custom transformation rule to open Activity Editor.

   Example: Figure 53 shows the Custom transformation we are using in this example. The source attribute is ObjClarify_QuoteSchedule.PriceProgExpireDate, and the destination attribute is ObjARInvoice.GLPostingDate.

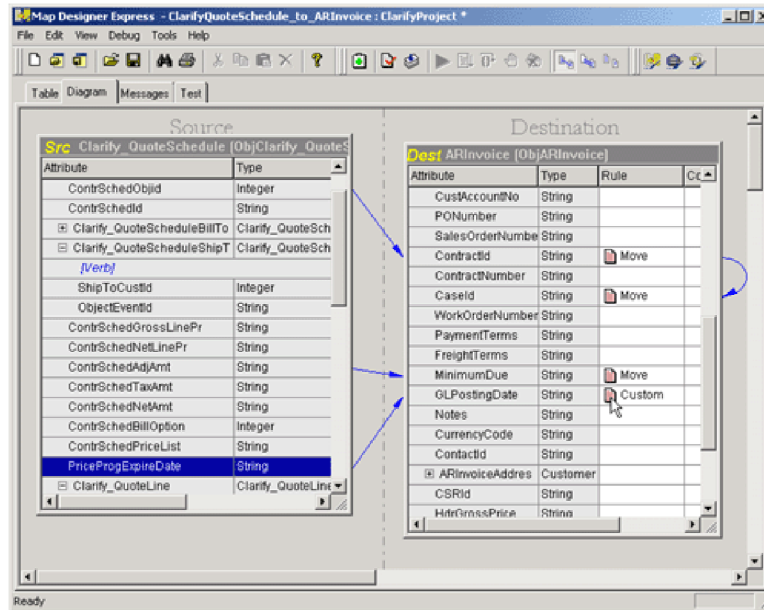


Figure 53. Custom transformation rule

   Result: Activity Editor opens.

   For more information on creating Custom and other transformations, see Chapter 2, "Creating maps," on page 13..

2. Select a category in the Library window (top left) to show the available function blocks in that category in the Content window (bottom left).

   Figure 54 shows the available functions blocks for the ″Date″ category; the source and destination attributes in our example are displayed as icons in the

graphical canvas.

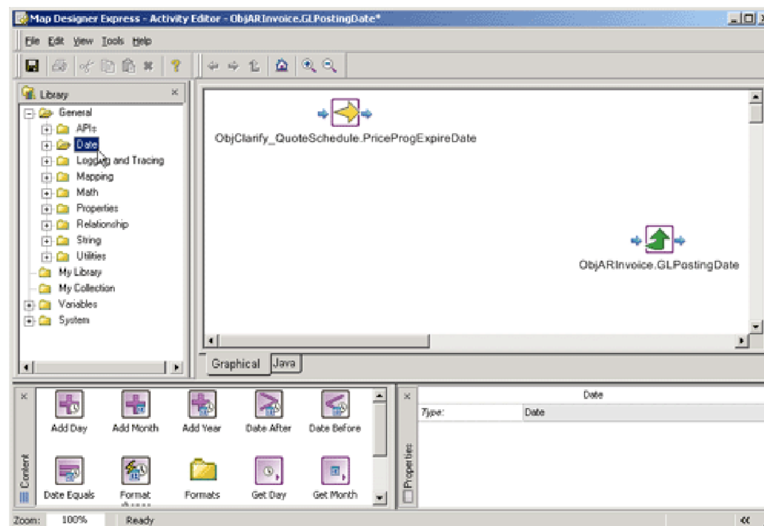


Figure 54. Function blocks in Date category and icons for source and destination attributes

3. To use any of the function blocks in the activity, drag the function block from the tree in the Library window and drop it onto the activity canvas; or alternatively, drag the icon from the Content window and drop it onto the activity canvas.

   Example: In our example, we want to change the date format of the source attribute from ″yyyyMMdd″ to ″yyyy.MM.dd G ′at′ HH:mm:ss z″ and assign it to the destination attribute; so we will drag-and-drop the Format Change function block in the Date category from the Content window onto the activity canvas, as shown in Figure 55..

   Note: A date formatted with ″yyyyMMdd″ looks like this: ″20030227″; a date formatted with ″yyyy.MM.dd G ′at′ HH:mm:ss z″ looks like this ″2003.02.27 AD at 00:00:00 PDT″.

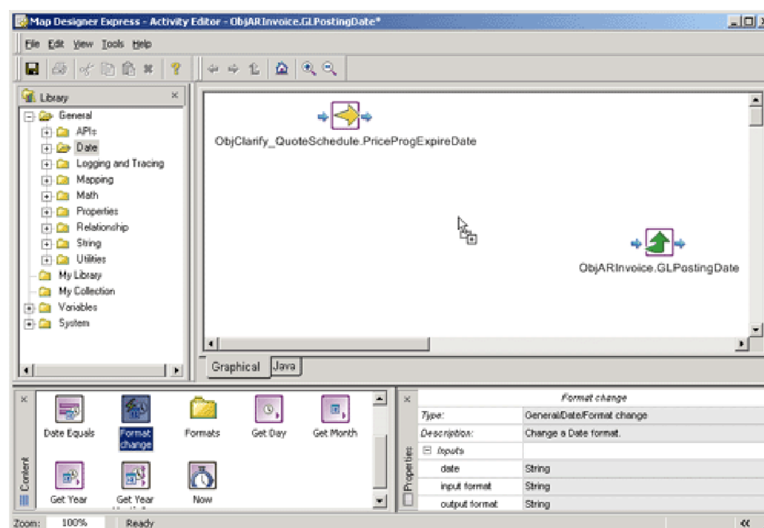


Figure 55. Dragging the Date Format Change function block

4. After you drop a function block on the activity canvas, you can move it around the canvas by selecting the function block icon and dragging and dropping it at the desired position. When the function block is in place, you are ready to connect the inputs and outputs of the function block to define the flow of execution.

   Example: In our example, we want to change the date format of the source attribute ObjClarify_QuoteSchedule.PriceProgExpireDate. We will do this by connecting the output of the port icon for ObjClarify_QuoteSchedule.PriceProgExpireDate to the date input of the Format Change function block. To do this, move the mouse cursor to the output of the icon of port ObjClarify_QuoteSchedule.PriceProgExpireDate.

   Result: The shape of the icon will change to an arrow to indicate that you can initiate a link at that point, as shown in Figure 56..

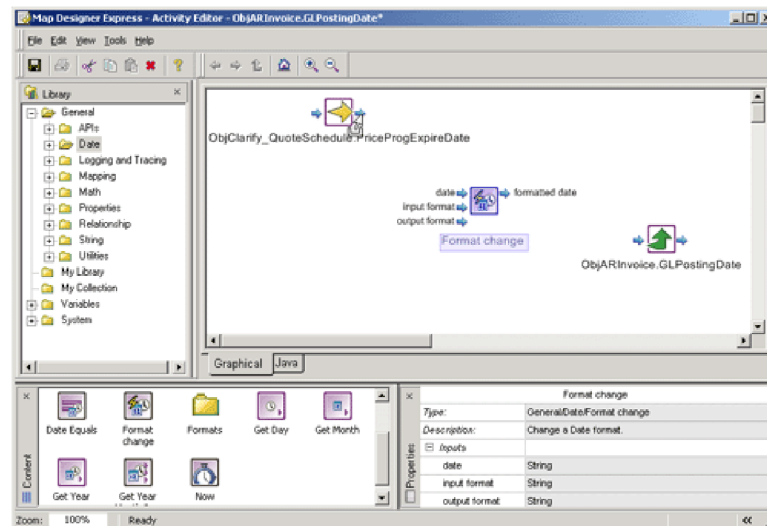


Figure 56. Cursor as arrow at output port of ObjClarify_QuoteSchedule.PriceProgExpireDate

5. When the mouse icon is changed to an arrow, hold down the mouse button and move the mouse to the date input of the Format Change function block, and release the mouse button. A connection link will be drawn to connect the input and outputs.

   To indicate that the result of the Format Change function block should be assigned to the destination attribute ObjARInvoice.GLPostingDate, repeat the same steps to drag-and-drop from the output of the Format Change function block to the input of the ObjARInvoice.GLPostingDate port icon. Figure 57

shows the connection links.

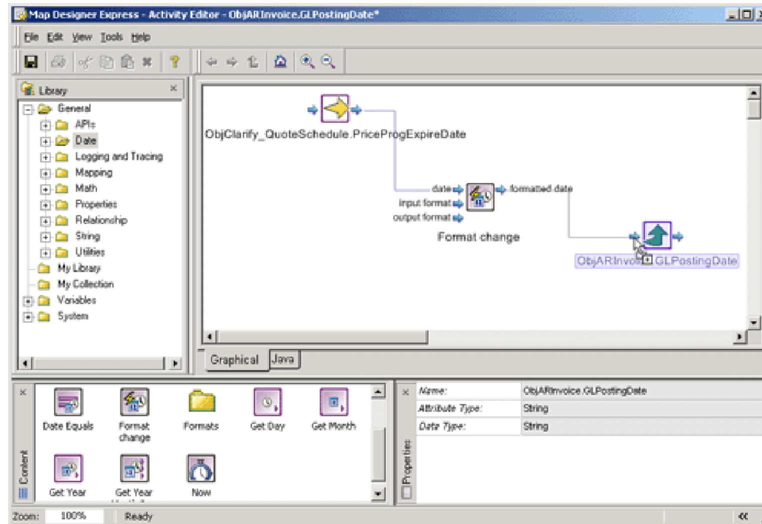


Figure 57. Date Format Change function block with connection links

Result: Now we have instructed the Format Change function block to take the input from the attribute ObjClarify_QuoteSchedule.PriceProgExpireDate, change its date format, and assign the result to the attribute ObjARInvoice.GLPostingDate. However, we still need to let the Format Change function block know what the original date format is and what resulting format we want.

6. **Example:** In our example, if the source attribute ObjClarify_QuoteSchedule.PriceProgExpireDate is in the date format of yyyMMDD (that is, 20030227), we can use the predefined Date Format function block yyyyMMdd. Drag-and-drop the yyyyMMdd function block onto the activity canvas and connect the format output of the yyyyMMdd function block to the input format of the Format Change function block.

   Result: This will specify that the input format of the date is in yyyyMMdd format, as shown in Figure 58..

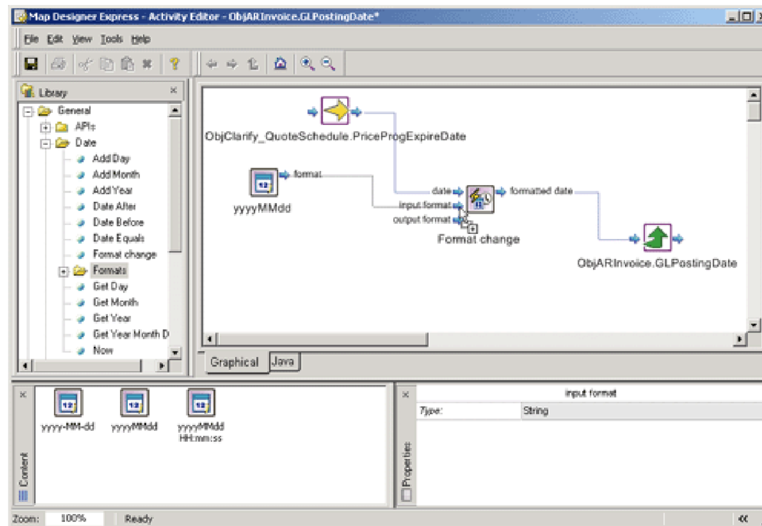


Figure 58. Input Date Format

7. Activity Editor provides three predefined Date formats: yyyyMMDD HH:mm:ss, yyyyMMDD, and yyyy-MM-dd. If the desired date format is not one of the three predefined formats, you can specify the date format you want by using a *Constant*. A *Constant* is a graphical component in which you enter text directly and use the text as input to function blocks or ports.

   Example: In our example, we want the Format Change function block to change the date format to yyyy.MM.dd G 'at' Hh"mm"ss z. This is not one of the predefined formats, so we will create a New Constant component in the activity canvas by dragging and dropping the *New Constant icon* (located under the System category) from the Content window to the activity canvas. Figure 59 shows the result of this action.

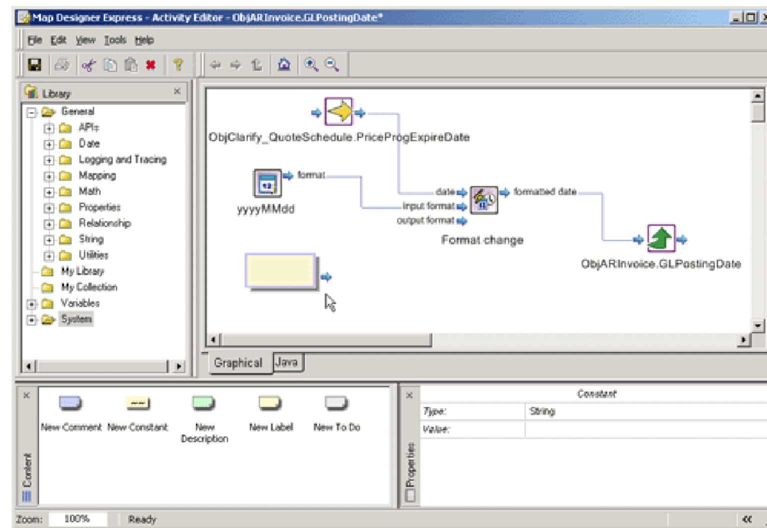


Figure 59. New Constant icon dropped on the activity canvas

8. To specify a constant with the value yyyy.MM.dd G 'at' Hh"mm"ss z, click the editable area of the Constant component in the activity canvas and enter the text `yyyy.MM.dd G 'at' Hh"mm"ss z`. By default, any Constant component will have the type String (shown in the Properties window when the Constant component is selected). However, you can change the type of the Constant by selecting the Constant and using the combo box in the Properties window.

Figure 60 shows the New Constant icon with the text value entered.

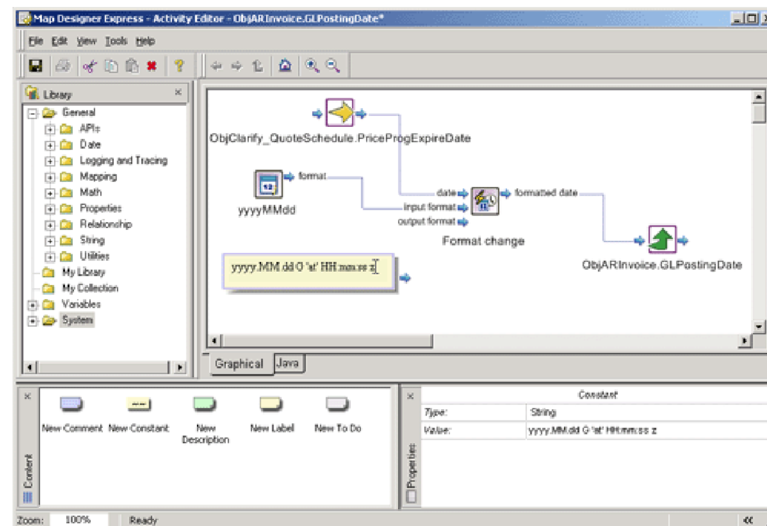


Figure 60. New Constant with text entered

9. To continue to specify that we want the output format of the Format Change function block as yyyy.MM.dd G 'at' Hh"mm"ss z, we define a connection link between the Constant component and the output format of the Format Change function block

   Result: We have completed the activity definition that will change the date format of the source attribute to a new date format and assign it to the destination attribute.

10. To add a comment or description to remind us later what this activity does, we can add a *Description* component to the activity and enter a description.

    Tip: Use the Context menu in the graphical canvas and select Add Description, or drag the *New Description icon* under the System folder in the Content window and drop it onto the activity canvas. Figure 61 shows how to add the Description component using the Context menu.

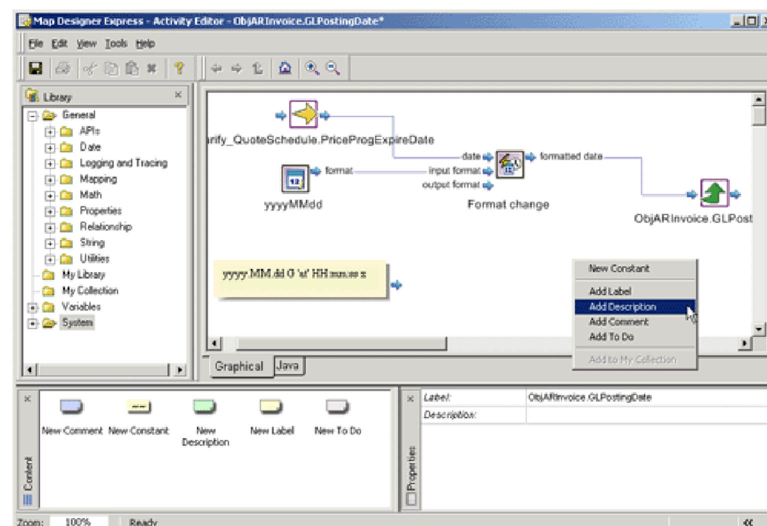


Figure 61. Adding a Description using the Context menu

Result: The Description component will be created in the graphical canvas.

11. Enter the description in the Description component by clicking on the editable area of the component and typing directly into the component. You can resize the Description by clicking and moving the lower right-hand corner of the Description component. Figure 62 shows adding the Description.

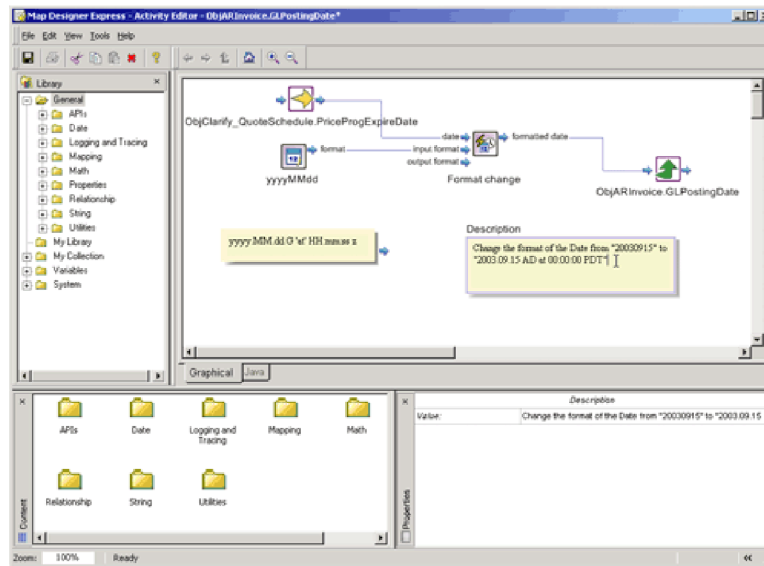


Figure 62. Adding the Description

12. Save the activity by selecting To Project or To File from the File-->Save submenu or by clicking the Save Map to Project or Save Map to File button in the Standard toolbar. Figure 63 shows saving the activity.

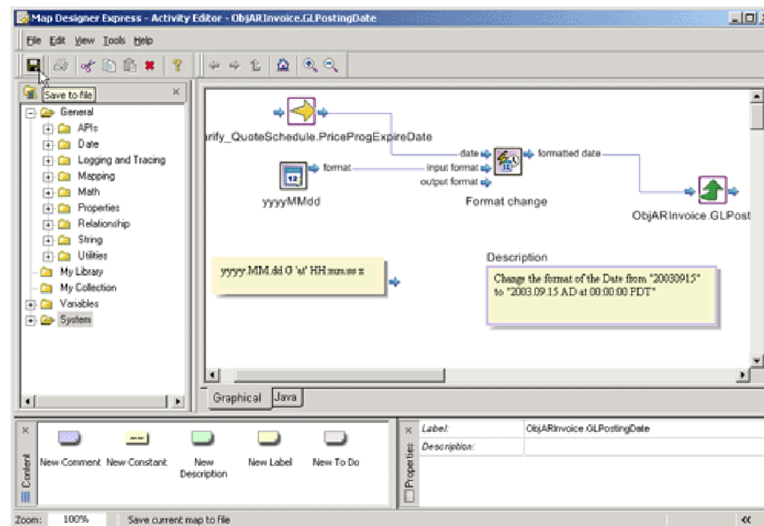


Figure 63. Saving the activity

Example 3 of using the Activity Editor

The following example illustrates using the *Static Lookup* relationship function block in Activity Editor.

In the WebSphere InterChange Server Express, a static lookup relationship normally consists of two or more relationship tables. For example, consider a system that consists of three end-applications, as shown in Figure 64.

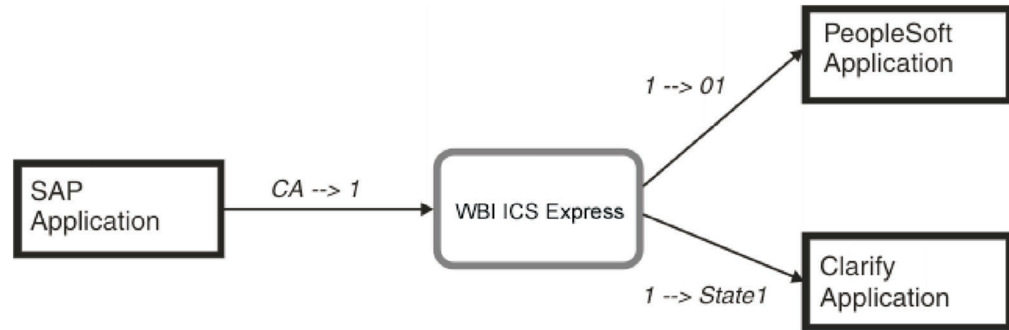


Figure 64. Static Lookup relationship with three end-applications

Each of these three applications has a different representation for ″State″ information, as shown in Table 53.

Table 53. Application-specific representation of state information

	SAP application	PeopleSoft application	Clarify application
California	CA	01	State1
Washington	WA	02	State2
Hawaii	HI	03	State3
Delaware	DE	04	State4

When state information is sent to the WebSphere business integration system from the SAP application, SAP specified-state code is sent to ICS. But when ICS needs to pass this information to other applications, the state information has to be converted to the format that the target application understands. In order to do this, the system needs a generic representation of the ″State″ information. With the generic representation, the system can process business logics in a generic, unified manner; and the generic representation will be converted to the application-specific format only when needed.

Thus, in the preceding example, we would create a static lookup relationship for doing this ″State″ conversion, with the application-specific data as WebSphere business integration-managed participants. With this setup, a generic ID is used to represent the state information in the WebSphere business integration system. Table 54 shows this representation.

Table 54. Generic representation of state information

	Generic ID	SAP application	PeopleSoft application	Clarify application
California	1	CA	01	State1
Washington	2	WA	02	State2
Hawaii	3	HI	03	State3
Delaware	4	DE	04	State4

Application-specific data is converted to the generic ID as it enters the ICS system, and the generic ID is converted to application-specific data as it exits the system.This data conversion is shown in Figure 65.

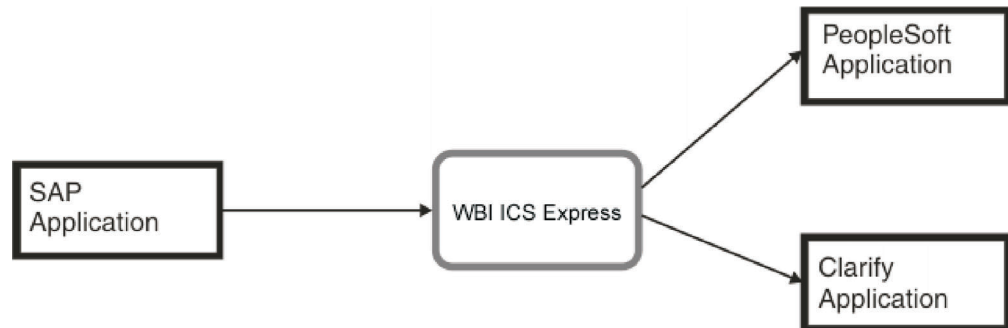


Figure 65. Data conversion from application-specific to generic to application-specific

The ID conversion is usually done in maps that convert application-specific business objects to generic business objects, or vice versa. For example in the SAP-to-Generic map, we would do a static lookup for the data ″CA″ and convert it to the generic representation that ICS understands, ″1″. And in the Generic-to-Clarify map, we would instead do a static lookup for the generic data ″1″ and convert it to ″State1″. In either map, only one static lookup is required.

Figure 66 shows how to use the Static Lookup function block to convert the SAP-specified state data to the ICS generic state data for processing in ICS.

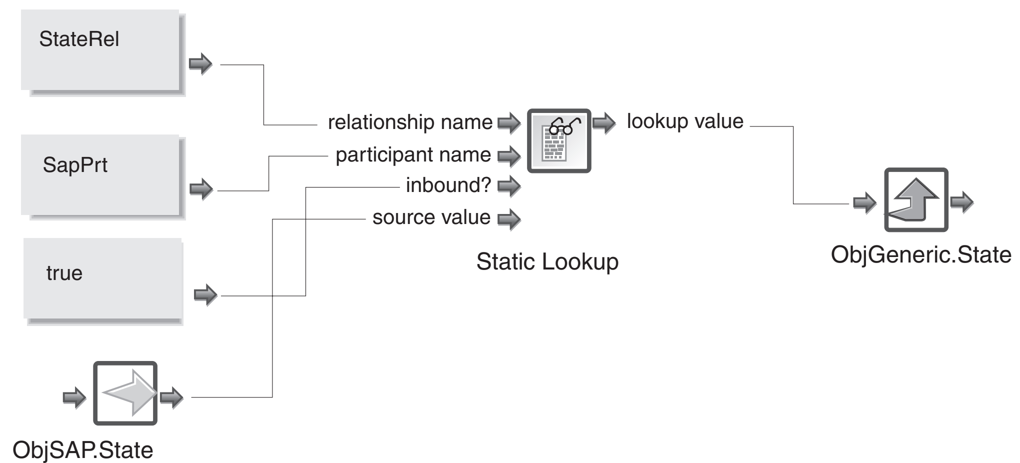


Figure 66. Using static lookup function block to convert SAP-specific state data to ICS-generic state data

Similarly, the Static Lookup function block is used to convert the ICS-generic state data to Clarify-specific state data in the Generic-to-Clarify map. This is shown in

Figure 67.

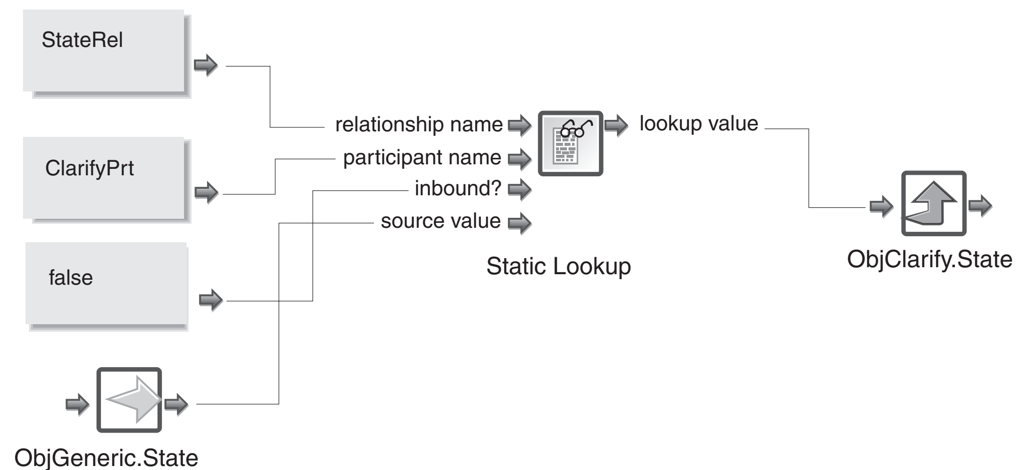


Figure 67. Using static lookup function block to convert ICS-generic state data to Clarify-specific state data

Normally, in a static lookup relationship, we convert application-specific data to generic data, or generic data to application-specific data. In these scenarios, only one Static Lookup function block is used. But in the special cases where you want to directly lookup a name-value pair, then two Static Lookup function blocks are required.

For more information on defining and using static relationships, see Chapter 7, "Creating relationship definitions," on page 167.

Working in Java view

If Map Designer Express opens the Activity Editor with an activity definition that contains only custom Java code, the Activity Editor displays the activity definition in Java view. Similar to Graphical view, the Activity Editor is available in Java view in two display modes: Design mode and Quick view mode.

- **Design mode:** In Design mode, the Java view of the Activity Editor contains the main Java WordPad for viewing and editing custom Java code to provide the definition for the activity. The WordPad is contained in a tabbed window area. In addition to the regular editing options in a WordPad (Cut, Copy, Paste, Delete, Select All, Undo, Redo), the Java WordPad provides syntax highlighting for the Java Programming language.

  By default, comments are green, string literals are pink, and keywords are blue.

  Tip: You can customize the syntax highlighting schemes in the Preference dialog.

Figure 68 shows the Java view in Design mode.

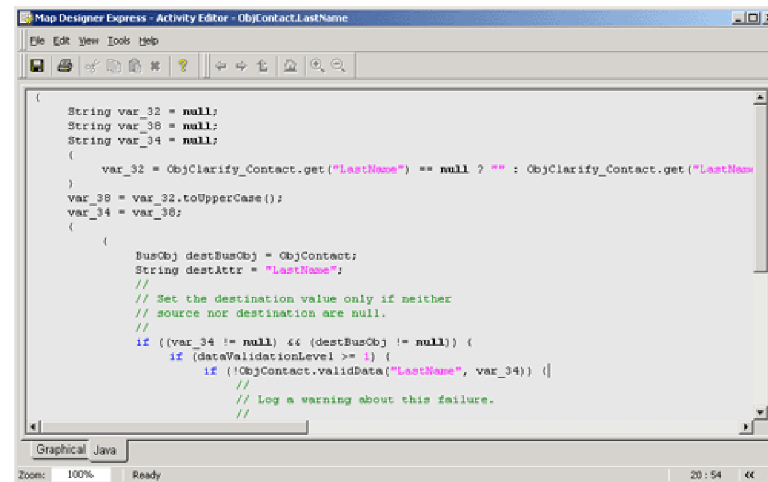


Figure 68. Java view in Design mode

- **Quick view mode:** In Quick view mode, the Java view only displays the WordPad. Figure 69 shows the Java view in Quick view mode.

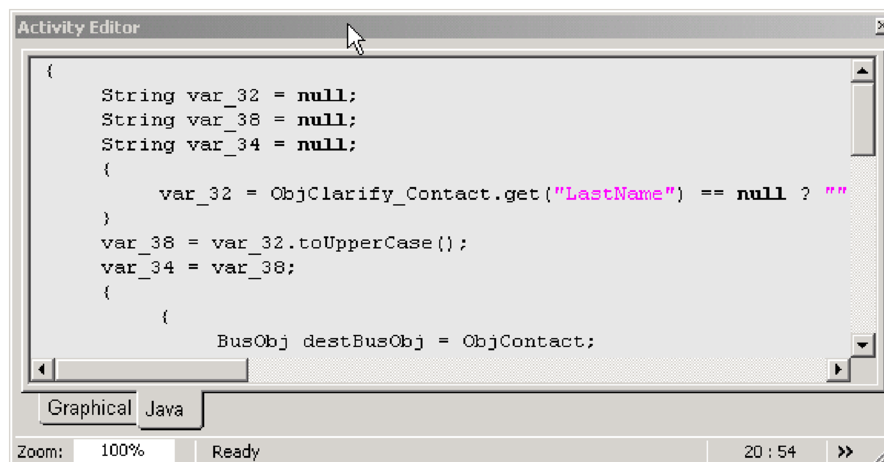


Figure 69. Java view in Quick view mode

Tip: To change from Quick view mode to Design mode, click the >> button on the status bar. If you do not see the >> button, resize the Quick view window horizontally until the button appears.

Importing Java packages to Interchange Server Express

In addition to using the standard function blocks that Activity Editor provides, Map Designer allows you to import your own Java library for use as function blocks in Activity Editor. Importing custom Jar libraries into activity settings will enable any public methods in the Jar library to be used as function blocks in Activity Editor.

Steps for importing Jar libraries as activity function blocks

Before you begin: You need to export your Java classes into a `.jar` file.

Perform the following steps to import a Jar library into Activity Editor:

1. In System Manager, open the Activity Settings view by clicking Window-->Show View-->Other... and selecting Activity Settings from the category WebSphere Business Integration system Manager.
2. Right-click BuildBlock Libraries and select Add Library. Figure 70 shows the Activity Settings view for adding a custom Jar library.

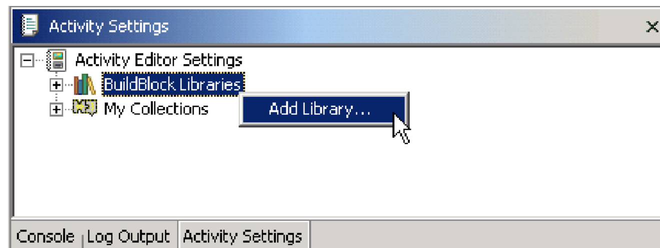


Figure 70. Activity Settings view

3. In the Open File dialog box, navigate to your custom `.jar` files and select Open.

   System Manager will try to import your custom `.jar` file for use as function blocks in Activity Editor. If the file is imported properly, the name of your custom `.jar` file will appear under BuildBlock Libraries in the Activity Settings view.

   Tip: After you import your custom `.jar` files into Activity Settings, when your maps and collaboration template compile in System Manager, the custom `.jar` file will automatically be included into the compile `CLASSPATH`. To prepare InterChange Server Express for compilation, make sure that its `CLASSPATH` includes your custom.jar file. For information on setting up Interchange Server Express for importing your custom `.jar` files, see "Importing third-party classes to Interchange Server Express" on page 139.

Result: When you open Activity Editor, the custom Jar library will be listed in the Library window under My Library in Activity Editor. By default, available custom function blocks are listed according to their package structure. You can use them in an activity the same way as standard function blocks.

Rule: After you change any settings in the Activity settings view, you must restart Map Designer Express for the change to take effect in Activity Editor.

Customizing display settings of custom Jar libraries

You can customize the display settings of the function blocks imported to Activity Editor, such as its name and icon, by changing the custom Jar library's properties. Perform the following steps to do this:

- Display the Properties window for the custom Jar library by right-clicking on your custom Jar library listed under BuildBlock Libraries in the Activity Settings view in System Manager.

  Result: When the Properties window for the custom Jar library is opened, it will list the available function blocks in this custom library in a tree structure on the right-hand side of the dialog. The available function blocks are listed as child nodes under the Java class and package that contain them.

For the Java package and classes, you can customize the display name of the entry and whether Activity Editor should display this Java package/class in the My Library tree structure by changing the check box ″Hide level in tree display.″ If this option is enabled, Activity Editor will not display this entry in the My Library subtree. This option is usually useful when the Java methods in your custom Jar library are in a Java class that is in a package many levels deep, and enabling this option can better organize your My Library subtree in Activity Editor.

Figure 71 shows the dialog for customizing the Jar library display.

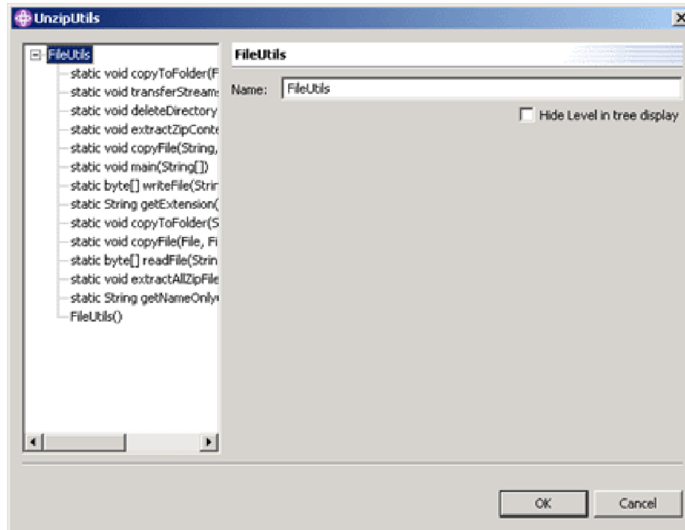


Figure 71. Properties dialog for customizing Jar library display

For those Java methods used as function blocks in Activity Editor, you can specify the function block display name, description, icon, and parameter’s display name in the Properties window. When you choose to import an icon for the function block, the icon that you choose will be copied into the Activity Settings folder and will be available for other function blocks in the same package to use.

Recommendation: If you choose to import an icon for your function block to use, the icon should be 32 pixels by 32 pixels in size and should be in `.bmp` format. The color depth of the icon can be up to 24-bit.

Figure 72 shows the Properties dialog for customizing the Jar library function block display.

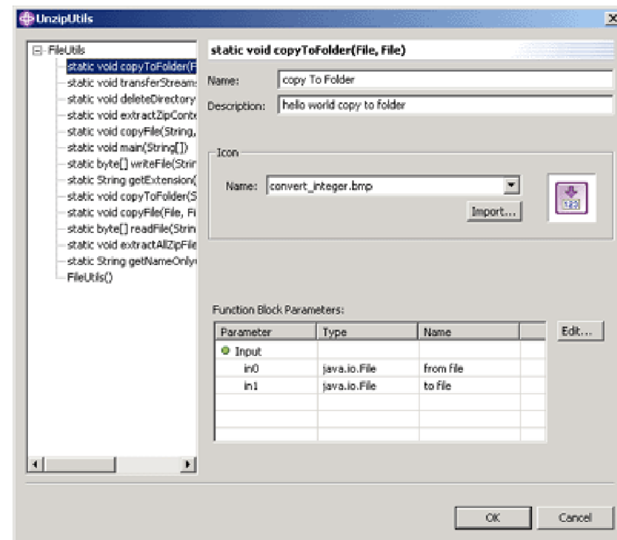


Figure 72. Properties dialog for customizing Jar library function block display

Rule: After you change any settings in the Activity settings view, you must restart Map Designer Express for the change to take effect in Activity Editor.

Importing third-party classes to Interchange Server Express

If the imported classes are in a third-party package rather than in the JDK, in order to set up the server compile, you must add them to the path of the imported classes in the `JCLASSES` variable.

Recommendation: You should use some mechanism to differentiate those classes in `JCLASSES` that are standard from those that are custom.

Example: You can create a new variable to hold only those custom classes and append this new variable to `JCLASSES`, by performing the following steps:

1. Create a new map property, such as one called `DEPENDENCIES`.
2. Place the `CwMacroUtils.jar` in its own directory.

   Example: Create a `dependencies` directory below the product directory and place the jar file in it.
3. Add the `dependencies` directory to the file used to start ICS (by default, `start_server.bat` or `CWSharedEnv.sh`), which is located in the `bin` directory below the product directory. For example, add the following entry for UNIX:

   ```
   set DEPENDENCIES=$ProductDir/dependencies/CwMacroUtils.jar
   ```

   Add the following entry for Windows:

   ```
   set DEPENDENCIES="%ProductDirS%"\dependencies\
   CwMacroUtils.jar
   ```
4. Add `DEPENDENCIES` to the `JCLASSES` entry:

   For UNIX, add:

   ```
   set JCLASSES=$JCLASSES:ExistingJarFiles:$DEPENDENCIES
   ```

   For Windows, add:

```
set JCLASSES=ExistingJarFiles;%DEPENDENCIES%
```

5. In each map that uses the classes, include the *PackageName.ClassName* specified in the `CwMacroUtils.jar` file.
6. Restart ICS to make the methods available to the maps.

Guidelines: When importing a custom class, you may get an error message indicating that the software could not find the custom class. If this occurs, check the following:

- Check that the custom class is part of a package. It is good programming practice for custom classes to be placed in a package. Make sure that the custom class code includes a correct package statement and that it is placed at the beginning of the source file, prior to any class or interface declarations.
- Be sure that you have updated the `CLASSPATH` environment variable to include the path to the package containing the custom class, or to the custom class itself if it is not in a package.

Example: When importing a custom class, you might create a folder called `%ProductDir%\lib\com\<ProductDir>\package`, where `package` is the name of your package. Then, place your custom class file under the folder you just created. Finally, in the `CLASSPATH` variable in the `start_server.bat` file, include the path `%ProductDir%\lib`.

Using variables

A variable is a placeholder for a value in the Java code. This section provides the following information about using variables in transformation code:

- "Using generated business object variables and attributes"
- "Using temporary variables" on page 142

Using generated business object variables and attributes

This section provides information about generating business object variables for the source and destination business objects.

Generating business object variables

When you add a business object to the map, Map Designer Express automatically generates the following:

- An instance name

  The instance name that Map Designer Express generates is a system-declared local variable that you can use to refer to this business object in the mapping code. It is prepended with the letters `Obj`, which is followed by the name of the business object definition.

  Example: If you add `Customer` to the map, its instance name is `ObjCustomer`. Map Designer Express generates an instance name for both the source and destination business objects.
- An index for the business object within a business object array (if the business object is multiple-cardinality)

  The business object index represents the order of this source or destination business object. The index number of the first source and destination business objects in a map is zero. Additional business objects take the next available index number, such as 1, 2, 3, and so on.

  When the map is executed, the index number represents the position of the business object in the array that is passed into the map (source business objects) or returned by the map (destination business objects).

Map Designer Express displays this information in the following locations:

- In the Business Objects tab of the Map Properties dialog

  Right-click the title bar of the business object window and select Properties from the Context menu. The Map Properties dialog appear with the Business Objects tab displaying and the selected business object highlighted in the list. This tab displays both the instance name and its index within the business object array (if the business object is multiple cardinality).
- In the Table tab—in the business object pane
- In the Diagram tab—in the title bar of the business object window in the following format:

  The title bar displays the instance name for the business object.

Note: You can specify whether Map Designer Express displays the names of the variables for the source and destination business objects with the option `Defining Map: show business object instance name`. By default, this option is enabled and Map Designer Express displays these variable names (`Obj`*`BusObj`*) in both the Table and Diagram tabs. When the option is disabled, Map Designer Express only displays the names of the source and destination business objects. You can change the setting of this option on the General tab of the Preferences dialog. For more information, see "Specifying General Preferences" on page 20.

You can modify these business object variables from the Business Objects tab of the Map Properties dialog (see Figure 73).

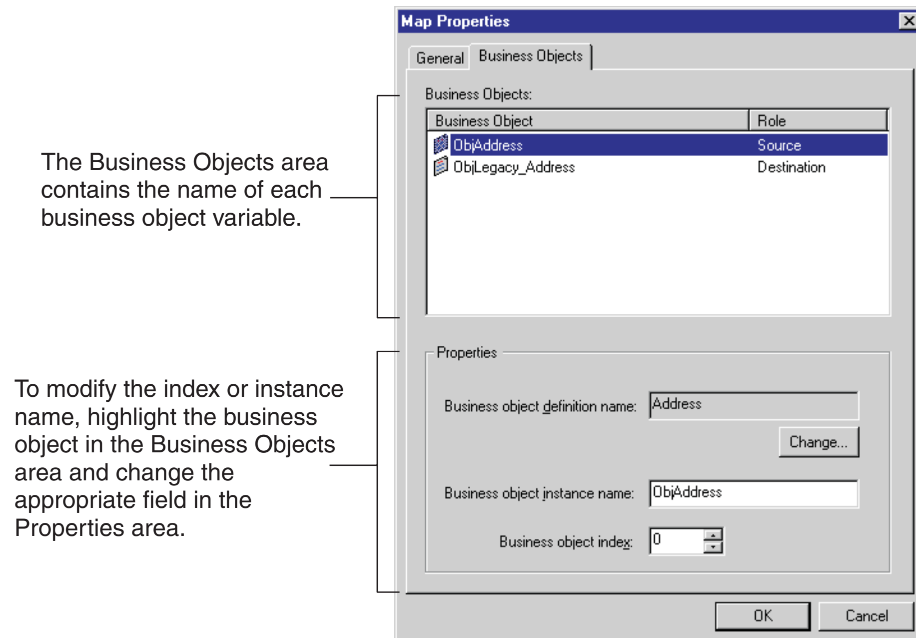


Figure 73. Business Objects Tab of the Map Properties Dialog

You can display the Business Objects tab of the Map Properties dialog in any of the following ways:

- From the Edit menu, select Map Properties. For information on other ways to display the Map Properties dialog, see "Providing map property information" on page 54. The General tab of the Map Properties dialog box appears. Click the Business Objects tab.

- From the Diagram tab, right-click the business object window and select Properties from the Context menu.

Using temporary variables

Map Designer Express lets you create temporary variables that can be accessed in transformation steps throughout the map; that is; temporary variables are global to the map. For example, you can calculate a value in one transformation step, store it in a temporary variable, and reference the variable in another transformation step. This is especially useful if a certain calculation is performed repeatedly; you can perform the calculation once, store the result in a temporary variable, and retrieve the value as needed (for example, with a Move transformation).

Temporary variables are defined within a temporary business object. You create a temporary business object from the Temporary tab of the Add Business Object dialog. To display the Add Business Object dialog, perform the following steps:

1. Select Add Business Object from the Edit menu.

   For information on other ways to display the Add Business Object dialog, see "From the Add Business Object dialog" on page 32.

   Result: The General tab of the Add Business Object Properties dialog box appears.
2. Click the Temporary tab. Figure 74 shows the Temporary tab of the Add Business Object dialog.

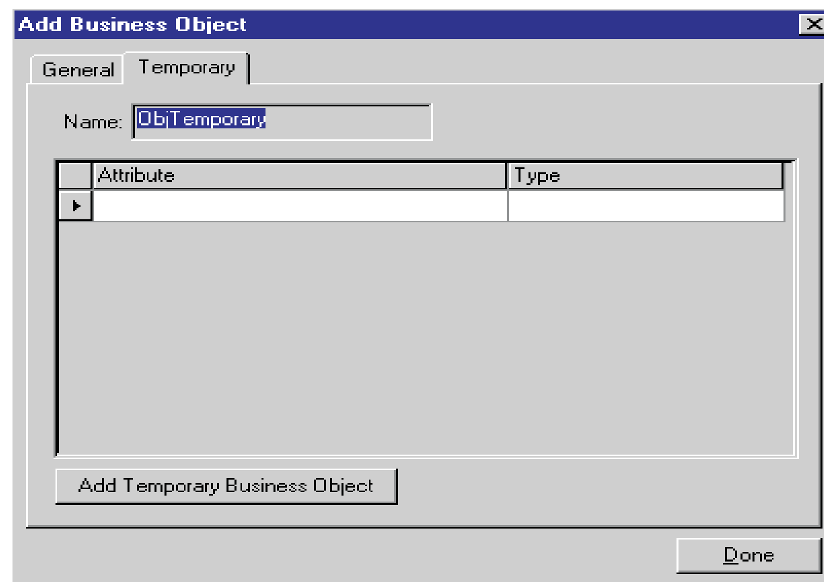


Figure 74. Temporary tab of the Add Business Object dialog

Through the Temporary tab of the Add Business Object dialog, you specify the temporary variables. To define a temporary variable:

1. Map Designer Express generates the temporary business object's name and displays it in the Name field. This field is read-only. The first generated name is `ObjTemporary`.
2. Click in the Attribute field.

   Result: A new row appears in the variables table. Enter the name of the temporary variable.

Note: Do *not* create two temporary variables with the same name.

3. Click the Type field and select the temporary variable's data type from the pull-down list.

   Note: To be compatible with the ICS data type scheme, all temporary variables have an internal type String. The data type specified in the Add Business Object dialog will affect only how the variable is initialized. If you want to write custom Java code to assign values to the temporary variable, the value has to be converted to a String first.
4. Repeat steps 2 and 3 for each of the temporary variables you need in the map.
5. Click the Add Temporary Business Object button.
6. You can either define another temporary business object or click Done to finish.

Once Map Designer Express creates the temporary business object, this business object appears in the Table and Diagram tabs with the map's other business objects, as follows:

- From the Table tab:
  - The business object pane adds a new area for the temporary business object. You can right-click the name of the temporary business object to get a Context menu that provides options to edit and delete this business object.
  - The temporary business object and its attributes appear in the combo boxes of the `Source Attribute` and `Dest. Attribute` columns in the attribute transformation table.
- From the Diagram tab, the map workspace adds a new business object window for the temporary business object.

  This Temporary business object window has many of the same characteristics as a business object window. Variables you create appear in the variables table just like attributes in a business object. This business object window provides a Rule and Comment column where you can add the temporary variable's transformation code and comment, respectively.

  You can right-click in the title bar of the Temporary business object window to get a Context menu that provides options to edit and delete this business object, as well as its properties.

  To specify a value for the variable, copy the value from a business object attribute to the variable by holding down the `Ctrl` key and dragging the attribute onto the variable name. You can also split and join attributes into a variable.

Note: A temporary business object also appears in the Business Object tab of the Map Properties dialog.

You use the temporary variable in a transformation step in this way:

- In the Diagram tab:
  - Click the row header (left-most) column of the temporary attribute.
  - Copy the variable value to an attribute by holding the `Ctrl` key and dragging the variable onto the attribute.

Important: Because a temporary variable is a global variable, you must explicitly initialize a temporary variable to `null` when using the Map Instance Reuse option. Otherwise, the value of the temporary variable from a previous execution of the map instance can incorrectly be used as the value of the temporary variable in subsequent executions of the same

map. When you do *not* use the Map Instance Reuse option, the InterChange Server Express system automatically initializes temporary variables between separate invocations of the map.

Reusing map instances

Typically, the map development system creates an instance of a map to process each transformation of data between the source and destination business objects. When the instance completes the handling of the transformation, the system frees up its resources. To reduce memory usage, the IBM system recycles an instance of a map instance by caching it and reusing it when the same type of map is instantiated at some later time. When the IBM system can recycle an existing map instance, it can avoid the overhead of map instantiation, thereby improving overall system performance and memory use.

The map development system automatically caches a map instance; that is, a map instance uses the Map Instance Reuse option by default.

Requirement: The Map Instance Reuse option imposes the following requirement on the map: If your map requires global variables, avoid initializing these global variables at declaration time. Instead, ensure that the global variables are always initialized at a map node, preferably the first transformation (attribute) node in a map.

Attention: A map containing global variables that are *not* initialized at the first transformation node cannot safely be recycled because the variable values in the cached map instance persist when the instance is reused. When the cached map instance is reused and begins execution, each global variable contains the value from the end of the previous use of the map instance.

If you cannot define your map so that it meets the preceding restrictions, you *must* disable the Map Instance Reuse option for this map. To disable this option, remove the check mark from the `Map instance reuse` box, which appears in the map's Map Properties window in System Manager. This window also allows you to specify the size of the map-instance pool. For more information on the Map Properties window of System Manager, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Note: Deploying the map to the server will not update the run-time instance. You can update the map properties dynamically from the server component management view by right-clicking on the map and selecting the properties from the Context menu. The changes will be automatically updated to the server.

Handling exceptions

An *exception* represents an occurrence that, if not handled explicitly within the map, stops the map's execution. During the execution of a map, run-time exceptions can occur. When you define a custom transformation rule, you can use the "Catch Error" function block to trap any run-time exception. Once you catch a particular exception, you can determine how to handle this exception.

Relationship exceptions

When using relationships in a map, several exceptions can occur. All of these exceptions are subclasses of `RelationshipRuntimeException`. If you are not concerned about the kind of exception, but simply want to catch them all, you can catch `RelationshipRuntimeException`. Otherwise, you can catch any of the following exceptions for specific cases:

- `RelationshipRuntimeDataAccessException`—thrown if a problem occurs while accessing the relationship database. You might catch this exception in any method call from the `Relationship` or `Participant` class.
- `RelationshipRuntimeDuplicateIdentityEntryException`—thrown if you try to add a participant to an identity relationship with the same relationship instance ID as an existing relationship instance. You might catch this exception in `addMyChildren()` and `create()` method calls.
- `RelationshipRuntimeUserErrorException`—is an abstract exception. It is thrown only if a `RelationshipRuntimeMetaDataErrorException` or `RelationshipRuntimeGeneralUserErrorException` occurs. You might catch this exception in any method call from the `Relationship` or `Participant` class during map development. Once the map is debugged, you can remove the handlers for this exception.
- `RelationshipRuntimeMetaDataErrorException`—thrown if an error occurs while manipulating the meta-data associated with participant instances, such as the relationship name or participant definition name. You might catch this exception in any method call that adds, modifies, or deletes participant instances.
- `RelationshipRuntimeGeneralUserErrorException`—thrown if there is an error in the run-time data supplied with a `Relationship` or `Participant` class method call.

  Example: The exception is thrown if you pass a business object of the wrong type to the `create()` method.

Figure 75 illustrates the relationship run-time exception hierarchy. Any exception you catch automatically catches those that are lower in the hierarchy. However, if an exception lower in the hierarchy is thrown, you cannot know exactly which one it is unless you catch it specifically.

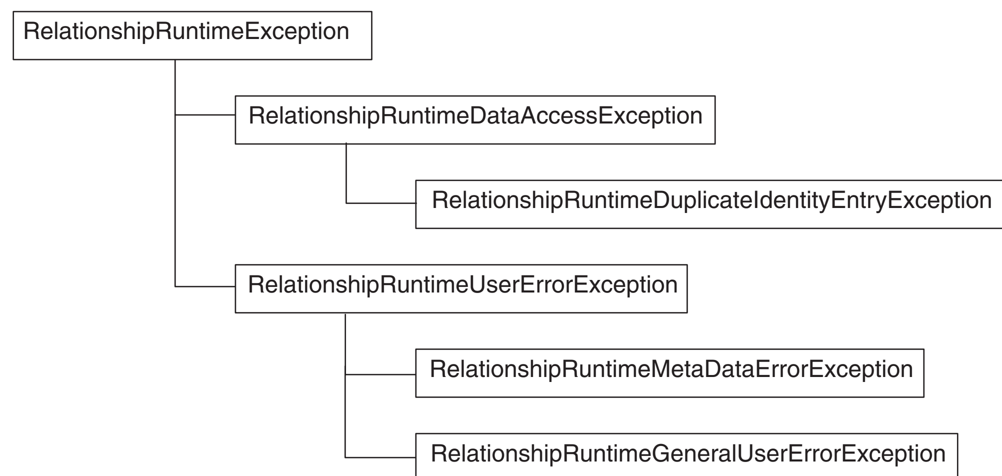


Figure 75. Relationship run-time exceptions

Example: If you catch `RelationshipRuntimeUserErrorException`, you automatically also catch `RelationshipRuntimeMetaDataErrorException` and `RelationshipRuntimeGeneralUserErrorException`. However, you cannot easily know which one of these was actually thrown, unless you test the exception with the `instance of` operator. The exception you choose to catch depends on how specific you want your exception handling to be.

Creating custom data validation levels

When values are mapped from one business object to another based on transformation code, incorrect data can result. The data validation feature checks each operation in a map and logs an error when data in the incoming business object cannot be transformed to data in the outgoing business object according to certain rules.

Example: Suppose that a map transforms a string value in the source business object to an integer value in the destination business object. This type conversion works properly when an incoming string value represents an integer (for example, "`1234`" represents the integer 1234). However, the conversion does not work properly if the string value does not represent an integer (for example, "ABCD" might indicate invalid data).

Coding a data validation level

The map development system defines data validation levels 0 and 1; levels 2 and greater are available for you to define. Table 55 summarizes the data validation levels:

Table 55. Data Validation Levels

Level	Description
0	Default; no data validation
1	IBM-defined data type checks
2 and greater	User-defined validation checks

Understanding map execution contexts

Each map instance executes within a specific *execution context* that is set by the connector controller. The Mapping API represents the map execution context with an instance of the `MapExeContext` class.

For every map that Map Designer Express generates, the map's execution context is accessible through a system-defined variable named `cwExecCtx`. You can reference this variable in the Variables folder in the Activity Editor.

Calling contexts

The *calling context* indicates the purpose for the current map execution. When transforming relationship attributes, you usually need to take actions based on the map's calling context. Table 56 lists the valid constants for calling contexts.

Table 56. Calling contexts

Calling-context constant	Description
EVENT_DELIVERY	The source business object(s) being mapped are event(s) from an application, sent from a connector to InterChange Server Express in response to a subscription request (event-triggered flow).
ACCESS_REQUEST	The source business object(s) being mapped are calls from an application, sent from an access client to InterChange Server Express (call-triggered flow).
ACCESS_RESPONSE	The source business object(s) being mapped are sent back to the access client in response to a subscription delivery request.
SERVICE_CALL_REQUEST	The source business object(s) being mapped are sent from InterChange Server Express to an application, through a connector.
SERVICE_CALL_RESPONSE	The source business object(s) being mapped are sent back to InterChange Server Express from an application as a response to a successful service call request.
SERVICE_CALL_FAILURE	The source business object(s) being mapped are sent back to InterChange Server Express from an application after a failed service call request.

You can reference these calling contexts as constants in the `MapExeContext` object that is available in every map that Map Designer Express creates.

Example: You reference the `SERVICE_CALL_REQUEST` calling context as `MapExeContext.SERVICE_CALL_REQUEST`.

Figure 76 illustrates when each of the calling context occurs in an event-triggered flow. Event-triggered flow is initiated when a connector sends an event to a collaboration in InterChange Server Express.

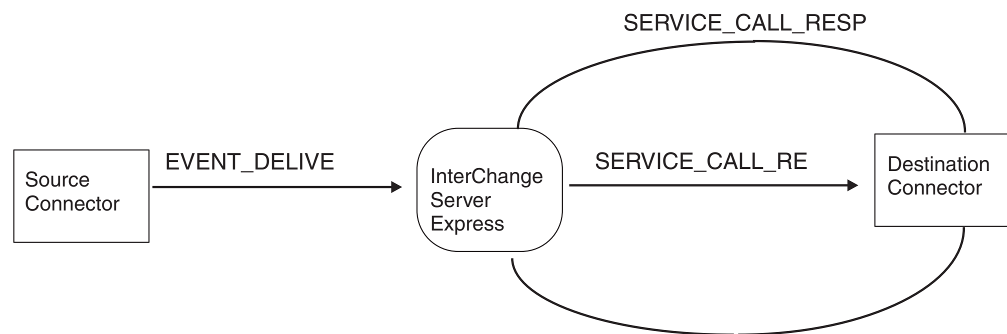


Figure 76. Calling contexts in an event-triggered flow

As Figure 76 shows, any mapping request coming from a connector to InterChange server (that is, a map from application-specific business object to generic business object) has a calling context of `EVENT_DELIVERY`. Any mapping request coming from InterChange server to a connector (that is, a map from generic business object to application-specific business object) has a calling context of `SERVICE_CALL_REQUEST`. Mapping requests sent by connectors in response to a collaboration's service call request can have contexts of `SERVICE_CALL_RESPONSE` or `SERVICE_CALL_FAILURE`.

Figure 77 illustrates when each of the calling contexts occurs in a call-triggered flow. Call-triggered flow is initiated when an access client sends a direct Server Access Interface call to a collaboration in InterChange Server Express.

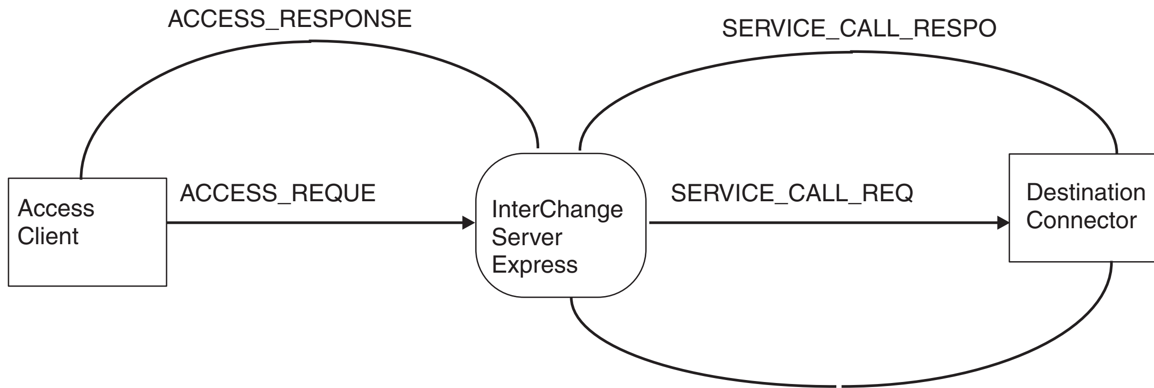


Figure 77. Calling contexts in a call-triggered flow

As Figure 77 shows, any mapping request coming from an access client to InterChange server (that is, a map from application-specific business object to generic business object) has a calling context of `ACCESS_REQUEST`. Any mapping request coming from InterChange Server Express to an access client (that is, a map from generic business object to application-specific business object) has a calling context of `ACCESS_RESPONSE`.

Original-request business objects

Another important part of the map's context is the *original-request business object*. This business object is the one that has initiated the map execution. Table 57 shows the calling contexts and the associated original-request business object.

Table 57. Calling contexts and their associated original-request business objects

Calling context	Original-request business object	Original-request business object from example
`EVENT_DELIVERY, ACCESS_REQUEST`	Application-specific business object that came in from the application	AppA-specific
`SERVICE_CALL_REQUEST, SERVICE_CALL_FAILURE`	Generic business object that was sent down from InterChange Server Express	Generic
`SERVICE_CALL_RESPONSE`	Generic business object that was sent down by the SERVICE_CALL_REQUEST	Generic
`ACCESS_RESPONSE`	Application-specific business object that came in from the access request initially	AppA-specific

For example, the *generic business object* is the original-request business object for maps that execute with a calling context of `SERVICE_CALL_RESPONSE`, `SERVICE_CALL_FAILURE`, or `SERVICE_CALL_REQUEST`. These maps use the generic business object to store relationship instance IDs for the relationship attributes being transformed. Having the relationship instance IDs is necessary for the map to look up the relationship instance and fill in the relevant participant data for newly created or updated objects.

Example: The following example illustrates how this might work in a customer synchronization scenario. Suppose you are using the system to keep data

synchronized between Application A and Application B. Both applications store customer data, and the customer ID attributes are managed using a relationship. For the purposes of this example, details about the collaborations and connectors involved are omitted.

When a new customer is added in Application A:

1. A map transforms an AppA-specific business object to a generic business object with a calling context of `EVENT_DELIVERY`.

   When transforming the customer ID attribute, the map creates a new relationship instance in the customer ID relationship table and inserts the new relationship instance ID into the customer ID attribute of the generic business object.
2. A map transforms the generic business object to a AppB-specific business object with a calling context of `SERVICE_CALL_REQUEST`.

   No changes occur to the relationship tables. Application B successfully adds the new customer to the application.
3. A map transforms the AppB-specific business object to a generic business object with a calling context of `SERVICE_CALL_RESPONSE`. The context for this map execution includes the generic business object generated in step 1.

   The reason for this execution is to fill in the new participant data for the relationship instance created in step 1. In this case, the new participant data is the customer ID for the new customer added to application B.

Figure 78 illustrates when the map execution for each step occurs for a call-triggered flow that successfully adds a new customer ID to Application B.

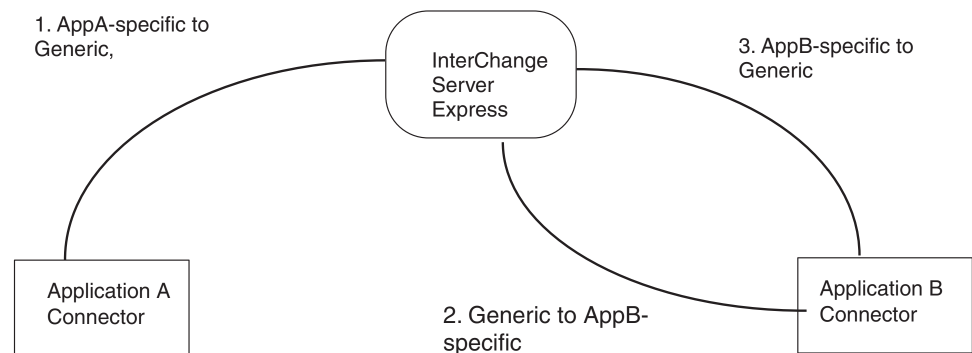


Figure 78. Example of Calling Contexts

Part 2. Relationships

© Copyright IBM Corp. 2003

Chapter 6. Introduction to Relationships

This chapter provides an overview of WebSphere business integration relationships and the relationship development process.

This chapter covers the following topics:

- "What is a relationship?" on page 153
- "Relationships: A closer look" on page 159
- "Overview of the relationship development process" on page 165

What is a relationship?

When attributes in a source and destination business object contain equivalent data that is represented differently, the transformation step employs a *relationship*. A relationship establishes an association between data from two or more business objects. Each business object is called a *participant* in the relationship.

The data that you typically transform using relationships are:
- ID numbers, such as a customer ID or product ID
- Other values represented as codes, such as country, currency, or marital status

Suppose application A uses sequential integers for customer IDs, and application B uses generated customer codes. `TashiCo` has a customer ID of `806` in application A and `A100` in application B. To transfer customer ID data between applications A and B, you can create a relationship among the application A customer business object, the generic customer business object, and the application B customer business object, based on the customer ID attributes.

This relationship establishes an association between customers from application A and application B, based on the key attributes of their customer business objects. In Figure 79, each box represents a participant in a relationship called `CustIden`.

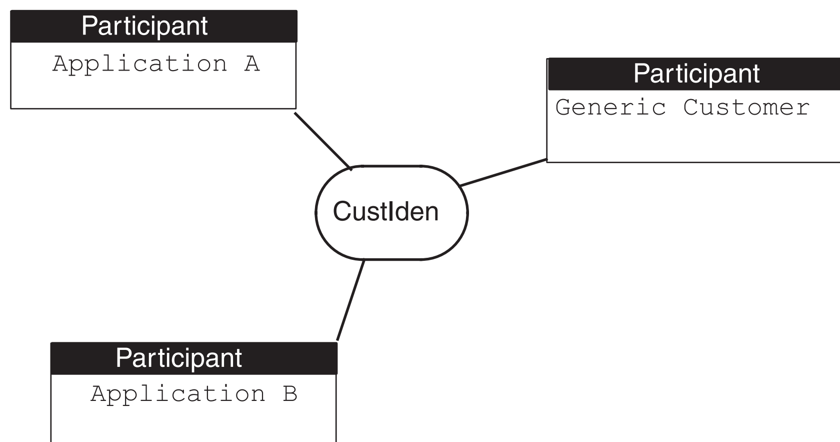


Figure 79. Relationship with three participants

Relationships are classified into the following categories based on the type of data in the participant and the number of instances of each participant that can be related:

© Copyright IBM Corp. 2003

- A *lookup relationship* establishes an association between data, such as attributes in business objects. The data can be related on a *one-to-one*, *one-to-many*, or *many-to-many* basis. Lookup relationships typically transform non-key attributes whose values are represented with codes, such as marital status or currency code. Use a lookup relationship if these attribute values are static; that is, new values are not often added or existing values removed.
- An *identity relationship* establishes an association between business objects or other data on a *one-to-one basis*. For each relationship instance, there can be only one instance of each participant. Identity relationships typically transform the key attributes of business objects, such as ID numbers and product codes. The relationship in Figure 79 is an example of an identity relationship. Use an identity relationship if key values are dynamic; that is, key values are frequently added or existing values are removed.
- A *non-identity relationship* establishes an association between business objects or other data on a *one-to-many* or *many-to-many* basis. For each relationship instance, there can be one or more instances of each participant. An example of a non-identity relationship is an RMA-to-Order transformation, in which a single RMA (Return Materials Authorization) business object can yield one or more Order business objects.

Lookup relationships

A *lookup relationship* relates two pieces of non-key data. For example, in a `Clarify_Site` to `Customer` map, you might transform attributes whose values are represented by codes or abbreviations, such as `SiteStatus`, using a lookup relationship. In a lookup relationship, there is one participant for each application-specific business object.

The `CustLkUp` relationship in Figure 80 establishes a lookup relationship between customer status codes from Clarify and SAP applications. Each box represents a participant in the `CustLkUp` lookup relationship. Notice that this relationship has two participants, one for each application-specific business object.

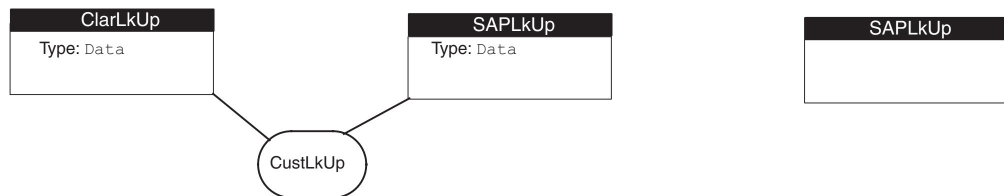


Figure 80. CustLkUp lookup relationship definition

Note: Because a lookup relationship does not indicate which attributes are being related, its participants use a special type called `Data`. For more information, see "Participant type" on page 164.

Suppose that the Clarify application represents an inactive customer with a site status of `Inactive` while in SAP the corresponding value is `05`. Although these customer status codes are different, they represent the same status, as Figure 81 shows.

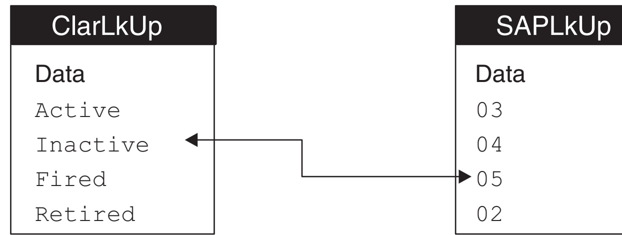


Figure 81. Relationship data for the CustLkUp lookup relationship

Table 58 shows the steps needed to create a lookup relationship.

Table 58. Steps for creating a lookup relationship

Creation step		For more information
1.	Define a lookup relationship in Relationship Designer Express.	"Defining lookup relationships" on page 176
2.	Customize mapping code to maintain the lookup relationship.	"Using lookup relationships" on page 188
3.	Test the lookup relationship to verify that it is implemented correctly.	"Testing a lookup relationship" on page 83

Identity relationships

An *identity relationship* establishes an association between business objects or other data on a *one-to-one* basis. To maintain a one-to-one relationship, each business object must have a key; that is, the object contains at least one attribute (a *key attribute*) whose value uniquely identities the object. If both business objects contain a key, they can participate in an identity relationship.

The WebSphere business integration system supports the following kinds of identity relationships:

- "Simple identity relationships"
- "Composite identity relationships" on page 157

Both kinds of identity relationships involve relating business object attributes. Therefore, each participant in an identity relationship has a business object as its participant type. For more information on participant types, see "Participant type" on page 164.

Simple identity relationships

A *simple identity relationship* relates two business objects through a single key attribute; that is, each business object contains a single value that uniquely identifies the object. Suppose the `CustIden` relationship (see Figure 79) is further refined to establish an association between customers from the Clarify and SAP applications, based on the key attributes of their customer business objects. In Figure 82, each box represents a participant in this customer identity relationship. Notice that this relationship has a participant for each application-specific business object and the generic business object.

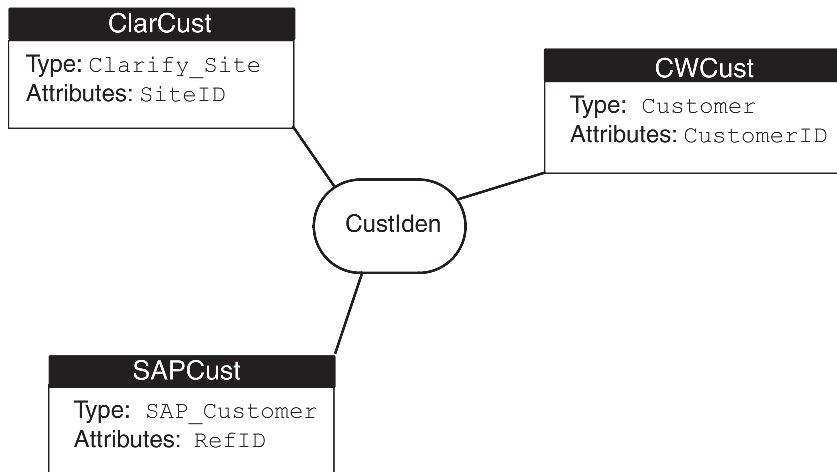


Figure 82. CustIden simple identity relationship definition

The TashiCo company is identified with a key value of A100 in the Clarify application while this same company is identified with a key value of 806 in the SAP application. Although these application IDs are different, they represent the same customer, as Figure 83 shows.

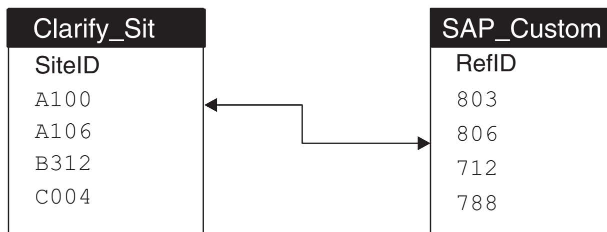


Figure 83. Relationship data for the custIden simple identity relationship

Therefore, the following maps use a simple identity relationship to maintain the transformations between the key attributes:

- The inbound maps (between the Clarify application-specific business object and the generic Customer business object) use a simple identity relationship to maintain the transformation between the SiteID attribute of the Clarify_Site business object and generic CustomerID attribute of the generic Customer business object.
- The outbound maps (between the generic Customer business object and the SAP application-specific business object) also use a simple identity relationship to maintain the transformation between the RefID attribute of the SAP_Customer business object and the generic CustomerID attribute of the generic Customer object.

Table 59 shows the steps needed to create a simple identity relationship.

Table 59. Steps for creating a simple identity relationship

Creation step		For more information
1.	Define a simple identity relationship in Relationship Designer Express.	"Defining identity relationships" on page 174
2.	Customize mapping code to maintain the simple identity relationship.	"Using simple identity relationships" on page 191

Table 59. Steps for creating a simple identity relationship (continued)

Creation step	For more information
3. Test the simple identity relationship to verify that it is implemented correctly.	"Testing an identity relationship" on page 80

Composite identity relationships

A *composite identity relationship* relates two business objects through a composite key. As the term "composite" indicates, a composite key is a key that consists of several attributes. Values for *all* attributes are needed to uniquely identify the object. A composite key consists of a unique key from a parent business object and a nonunique key from a child business object.

Suppose a particular order from TashiCo in the Clarify application is identified with a key value of 8765. This same order in the SAP application is identified with a key value of 0003411. Because these two order numbers uniquely identify the same order, their key attributes are related with a simple identity relationship. However, an order also contains order lines. If all participating applications identify these order lines with a unique value, a simple identity relationship can maintain their transformations.

However, it is often the case that an application uses only the line number to identify an order-line item. That is, each order contains a line item identified with 1, with any subsequent items numbered 2, 3, and so on. These line numbers do *not* uniquely identify the order-line items. To uniquely identify such items, the application uses a composite key that consists of the order number (from the parent order business object) and the line number (from the child order-line business object).

In Figure 84, the OrdrLine relationship establishes a relationship between order lines from the Clarify and SAP applications, based on their composite key attributes: the unique key attribute of their parent order business object combined with the order-line number in their child order-line business object. Each box represents a participant in the OrdrLine composite identity relationship. Notice that each participant has two attributes.

Figure 84. OrdrLine composite identity relationship definition

Suppose the Clarify application (represented by the participant `ClarOrder` in Figure 84) uses sequential integers to identify order-line items, while the SAP application uses the line number to identify these items. The Clarify application uniquely identifies each order-line item. Therefore, the maps between the Clarify application-specific business object and the generic `Order` business object (represented by the participant `CWOrder`) can use a simple identity relationship to maintain the transformation of the order-line items.

However, the SAP application (represented by the participant `SAPOrder`) identifies order-line items with their line number. Its items are not uniquely identified: every order contains a line item identified with 1, with any subsequent items numbered 2, 3, and so on. To uniquely identify the third order-line item of Order 0003411, you need to use a composite key, which includes both the order number (0003411) and the item number (3). Therefore, the maps between the SAP application-specific business object and the generic `Order` business object must use a composite identity relationship to maintain the transformation of the order-line items.

The third line item from the `TashiCo` order (`8765`) is identified in the Clarify application with the simple key value of `1171`. However, this same line item is identified in the SAP application with a composite key value of `0003411` (order number) and `3` (line number). Although these order lines are identified differently, they represent the same order line item, as Figure 85 shows.

Clarify_SF AQuote
QuoteObjId Clarify_QuoteLine
8764
8765 1168
8765 1169
8765 1170
8766 1171
8766 1172
1173

SAP_Order
OrderId SAP_OrderLineItem
0003409
0003410 1
0003410 1
0003411 2
0003411 1
0003411 2
3

Figure 85. Relationship data for the OrdrLine composite identity relationship

Table 60 shows the steps needed to create a composite identity relationship.

Table 60. Steps for creating a composite identity relationship

Creation step		For more information
1.	Define a composite identity relationship in Relationship Designer Express.	"Defining identity relationships" on page 174
2.	Customize mapping code to maintain the composite identity relationship.	"Using composite identity relationships" on page 202
3.	Test the composite identity relationship to verify that it is implemented correctly.	"Testing an identity relationship" on page 80

Relationships: A closer look

To understand the types of relationships that the WebSphere business integration system supports, you must understand how IBM implements the following concepts:

- "Relationships"
- "Participants" on page 163

Relationships

As Table 61 shows, a relationship is a two-part entity, consisting of a repository entity and a run-time object.

Table 61. Parts of a relationship

Repository entity	Run-time object
Relationship definition	Relationship instance

Relationship definition

You define a relationship to the WebSphere business integration system with a *relationship definition*. Relationship definitions identify each participant and specify how the participants are related. In Figure 79, `CustIden` is the relationship definition and it includes information about the three participants, Application A, Application B, and Generic Customer.

The system stores relationship definitions in the repository. The Relationship Designer Express tool provides dialogs to help you create the relationship definitions. Using this tool, you also store the completed relationship definition in the repository.

Tip: For more information on how to use Relationship Designer Express to create relationship definitions, see "Customizing the main window" on page 170.

The relationship definition provides the following information about the relationship:
- The relationship name
- The name of the relationship database

Relationship definition name: A relationship definition is simply a template, or description, of the relationship; it is *not* an actual business object. Therefore, the name of the relationship definition should *not* be the name of the associated business object.

Relationship database: The *relationship database* holds the relationship tables for a relationship. The relationship uses these relationship tables to keep track of the related application-specific values. For more information, see "Relationship tables" on page 161.

To access the relationship database at run time, the system must have the following information:
- The type of database management system (DBMS) that manages the relationship database
- The name and password of the user account that accesses the relationship database
- The location of the relationship database

By default, the relationship database is the WebSphere business integration system repository; that is, Relationship Designer Express creates all relationship tables in the repository. Relationship Designer Express allows you to specify the location of relationship tables in either of the following ways:
- Change the default location of relationship databases of *every* relationship.

  For more information, see "Global default settings" on page 182.
- Customize the location of each relationship's tables as part of the process of creating a relationship definition.

  For more information, see "Advanced settings for relationship definitions" on page 179.

Relationship instance

The relationship definition is a template for the run-time instantiation of the relationship, called the *relationship instance*. During map execution, the system creates instances of the relationship based on the relationship definition and using the values from the actual business objects being transformed.

For example, the relationship data for the `CustLkUp` lookup relationship (see Figure 81) shows that a customer status of `Inactive` in a Clarify application is the same as a customer status of `05` in an SAP application. Although these status codes are different, they represent the same customer status and therefore are in the same relationship instance, as Figure 86 shows.

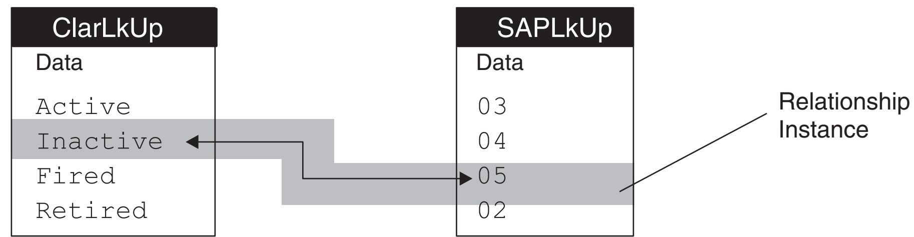


Figure 86. One Relationship instance for the CustLkUp relationship

A relationship instance is represented in the Mapping API by an instance of the `Relationship` or `IdentityRelationship` class.

To locate a relationship instance, the system requires the following information:

- A *relationship table* to identify which table contains the relationship instances for a particular participant
- A *relationship instance ID* to identify the actual relationship instance within the relationship table

Relationship tables: A *relationship table* is a database table that holds the relationship run-time data for one participant in a relationship. InterChange Server Express stores relationship instances in relationship tables, with one table (sometimes called a *participant table*) storing information for one participant in the relationship. For example, for the `CustLkUp` lookup relationship in Figure 80, InterChange Server Express requires two participant tables, as shown in Figure 86.

When you create a relationship definition, Relationship Designer Express automatically creates the table schemas that the relationship requires; that is, it creates the relationship tables with the necessary columns for each participant. At run time, these tables hold the data for the relationship instances.

Note: For an identity relationship, InterChange Server Express automatically populates the relationship tables. For a lookup relationship, you must populate the relationship tables with data. For more information, see "Populating lookup tables with data" on page 189.

To access a relationship table at run time, the system must have the following information:

- The name of the relationship table

  Because a relationship table is associated with a participant, the name of this table is defined as part of the participant definition. By default, any relationship table has a name of the form:

  `RelationshipDefName_ParticipantDefName`

  Relationship Designer Express allows you to customize the name of a relationship table as part of the process of creating a participant definition.

  For more information, see "Advanced settings for participant definitions" on page 180.
- The name of the database that contains the relationship table

  The name of the relationship database is set as part of the relationship definition. By default, the relationship database is the system repository. For more information, see "Advanced settings for relationship definitions" on page 179.

In map-transformation steps, relationship tables are managed using methods in the `Relationship`, `IdentityRelationship`, and `Participant` classes. Some Mapping API methods automatically manage relationship tables. You can also explicitly access these relationship tables to obtain this relationship data.

Relationship instance ID: The WebSphere business integration system uniquely identifies each relationship instance by assigning it a unique integer value, called a *relationship instance ID*. This instance ID allows the system to correlate the participant values. In general, given any participant in a relationship, you can retrieve the data for any other participant in the relationship by specifying the relationship instance ID.

For example, for the relationship between customer status codes of a Clarify application and an SAP application, the WebSphere business integration system assigns a relationship instance ID to each relationship instance of the lookup relationship. Figure 87 shows how instance ID `47` associates the Clarify customer status of `Inactive` with the SAP customer status value of `05`. Notice that this relationship is basically the same as the one in Figure 86, with the addition of the relationship instance ID.

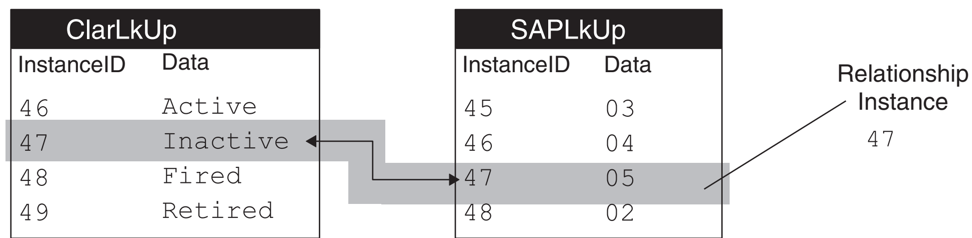


Figure 87. A lookup relationship with relationship instance IDs

Figure 87 shows the use of relationship instance IDs in a lookup relationship. Instance ID `47` associates the two application-specific participants, `ClarLkUp` and `SAPLkUp`.

The WebSphere business integration system also uses a relationship instance ID for the relationship between participants in an identity relationship. In the `CustIden` relationship (see Figure 82), this instance ID associates the customer IDs stored in the `SiteID` attribute of the `Clarify_Site` business object, the `CustomerID` attribute of the generic `Customer` business object, and the `RefID` attribute of the `SAP_Customer` business object. Figure 88 shows how the relationship instance data for each participant of the `CustIden` relationship is associated using the relationship instance ID.

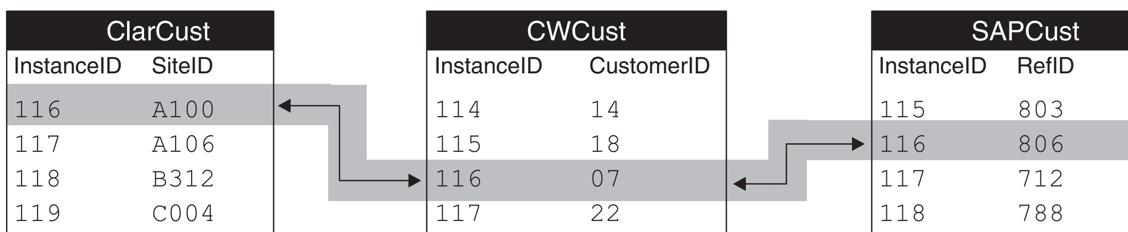


Figure 88. A customer identity relationship with relationship instance IDs

In Figure 88, the relationship table for the `CWCust` participant is included for clarity, though the table is not strictly necessary. In fact, relationship tables for the

participant representing the generic business object in any relationship are necessary *only* if you want to generate a generic ID for the associated attribute in the generic business object. The relationship in Figure 88 generates a generic ID (07) for the CustomerID attribute in the generic Customer business object.

You can simplify your relationship definition and increase performance by eliminating the relationship tables for the participant that represents the generic business object. You do this by checking the managed option for the participant when you create the relationship definition. See "Advanced settings for participant definitions" on page 180 for more information about this setting.

Figure 89 shows how relationship instance data is associated in the CustIden relationship when the Managed setting is specified for the CWCust participant.

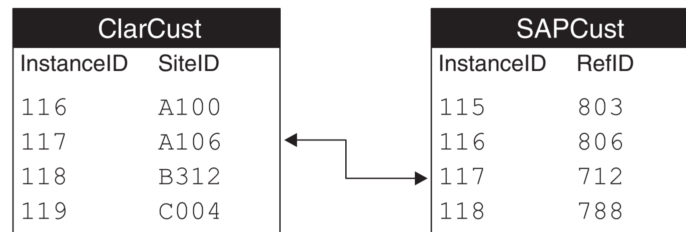


Figure 89. An identity relationship Instance with no generic table

The WebSphere business integration system stores the relationship instance ID in the relationship table for each participant. As Figure 87 through Figure 89 show, each relationship table in a relationship has a column that contains the relationship instance ID. ICS Express automatically creates the instance ID column when it creates the table schema.

Participants

A relationship contains *participants*, which describe the entities participating in the relationship. As Table 62 shows, a participant is a two-part entity, consisting of a repository definition and a run-time object.

Table 62. Parts of a participant

Repository entity	Run-time object
Participant definition	Participant instance

Participant definitions

The relationship definition contains a list of *participant definitions*. For instance, the CustIden relationship definition in Figure 82 associates customer business objects in Clarify and SAP and contains these participant definitions: SAPCust, CWCust, and ClarCust.

The WebSphere business integration system stores participant definitions in the repository. The Relationship Designer Express tool provides dialogs to help you create the participant definitions. Using this tool, you also store the completed participant definition in the repository.

The participant definition provides the following information about the participant:

- The participant name
- The participant type

- The name of the participant table and stored procedures

Participant definition name: A participant definition is simply a template, or description, of the participant; it is *not* an actual business object. Therefore, the name of the participant definition should *not* be the name of the associated business object.

Participant type: Like the attributes in a business object definition, the participants in a relationship definition have an associated type. The participant type specifies the kind of data associated with instances of the participant. The participant type can be one of the following:

- The name of a business object definition

  Relationships with participants of this type establish an association between entire business objects. In this case, you specify the attributes of the business object that relate the participant to the other participants in the relationship. The attributes you choose, usually the key attributes of the business object, become the *participant instance identifiers*.
- The word `Data`.

  In the participant definition, `Data` represents a supported attribute data type, such as `String`, `long`, `int`, `double`, `float`, or `boolean`. You specify `Data` as the type for participants in relationships that establish associations between specific attributes in business objects. Participants in lookup relationships have a participant type of `Data`.

For information on how to define the type of a participant, see "Creating relationship definitions" on page 173.

Participant table and stored procedures: For every participant, ICS Express creates the following database entities:

- A participant table to hold the relationship instance IDs and the associated participant's application-specific value
- Stored procedures to perform Retrieve (Select), Insert, Delete, and Update operations on the participant table

By default, Relationship Designer Express assigns names of the following form to the participant's table and stored procedure: *`RelName_ParticipantName_X`*, where *`RelName`* is the name of the relationship definition, *`ParticipantName`* is the name of the participant definition, and *`X`* is `T` for the participant table or `SP` for the stored procedure. By default, Relationship Designer Express creates the relationship tables in the WebSphere business integration system repository.

Relationship Designer Express allows you to customize the names of the participant table and stored procedures. For more information on naming the participant table and stored procedures, see "Advanced settings for participant definitions" on page 180.

Participant instances

The participant definition is a template for the run-time instantiation of the participant, called the *participant instance*. During map execution, the WebSphere business integration system creates instances of the participant based on the participant definition and the attribute values from the actual business objects being transformed.

The WebSphere business integration system stores participant instances as a column in the participant's relationship table. For example, for the `CustIden`

relationship in Figure 82, the `ClarCust` participant has a column called `SiteID` in its participant table to hold the values of its participant instances. The `SAPCust` participant has a `RefID` column in its participant table to hold the values of its participant instances.

Each participant instance contains the following information:
- Name of the relationship definition
- Relationship instance ID
- Name of the participant definition
- Data to associate with the participant

A participant instance is represented in the Mapping API by an instance of the `Participant` class.

Overview of the relationship development process

A relationship in the WebSphere business integration system is a two-part entity:
- A relationship definition, stored in the repository, to define the participants
- Code within a map to implement the relationship by accessing the relationship tables

To define a relationship in the WebSphere business integration system, you must perform the following basic steps:
1. Determine the type of relationship you need.
2. Within Relationship Designer Express, define a relationship definition and define the composite participants.
3. Within Map Designer Express, customize the transformation rule, if necessary, to maintain the relationship.
4. Recompile the affected maps.
5. Deploy the relationships and maps to InterChange Server Express with the Create Schema option.
6. Ensure that the relationship database(s) exists and is defined correctly within the relationship definition.
7. Populate relationship tables for any lookup relationships.
8. Optionally, populate other relationship tables with test data for the testing phase.
9. For each map, start all relationships in the map.
10. Test the relationship with the Test Connector. Be sure to set the appropriate calling context as part of each of the tests.

Figure 90 provides a visual overview of the relationship development process and provides a quick reference to chapters where you can find information on specific topics. Note that if a team of people is available for map development, the major tasks of developing a map can be done in parallel by different members of the development team.

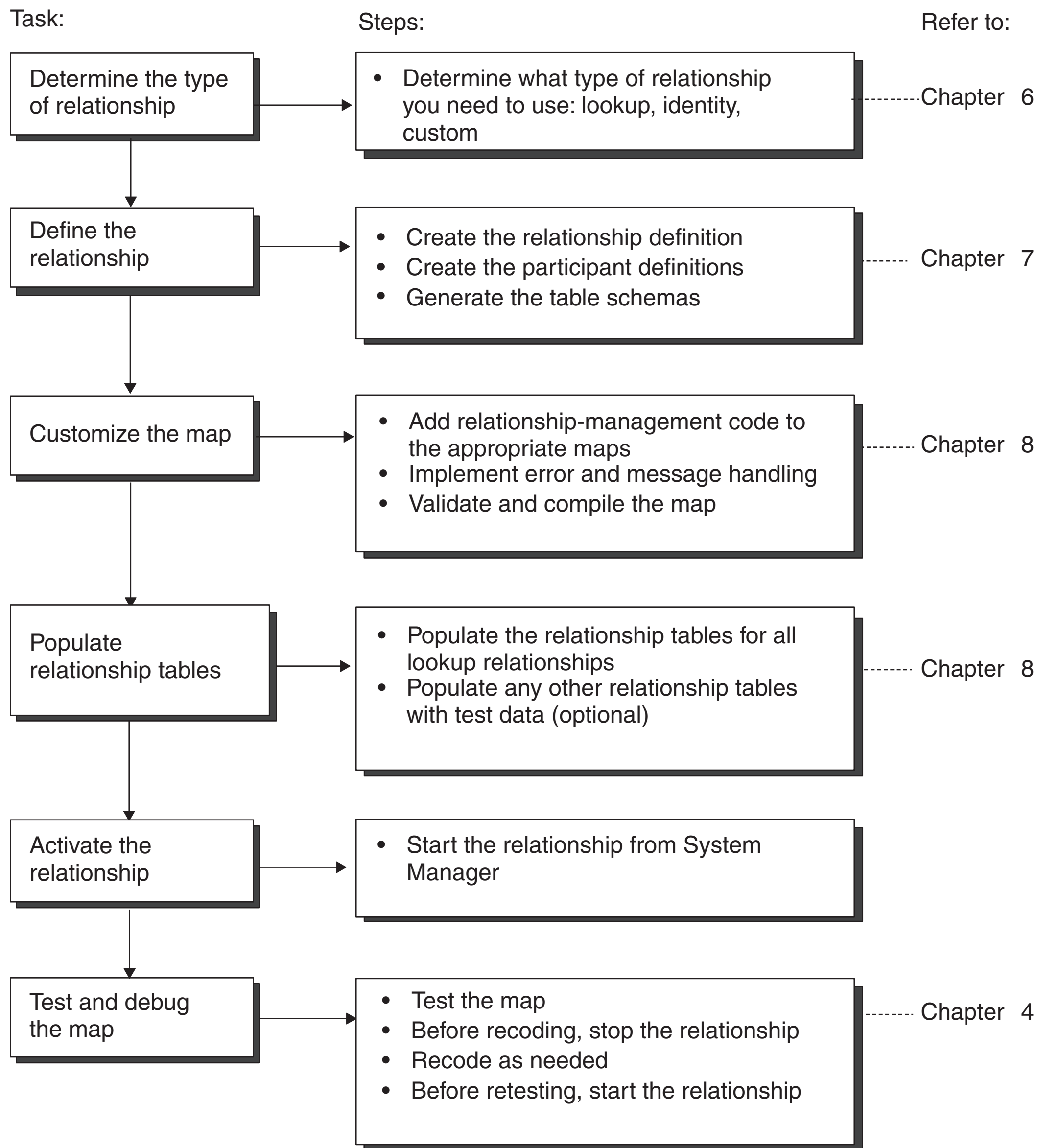


Figure 90. Overview of the Relationship Development Task

Chapter 7. Creating relationship definitions

This chapter describes how to create and modify relationship definitions using Relationship Designer Express. For background information on how the WebSphere business integration system uses relationships in mapping, see Chapter 6, “Introduction to Relationships,” on page 153. For help customizing relationships in maps, see Chapter 5, “Customizing a map,” on page 87.

This chapter covers the following topics:

- “Overview of Relationship Designer Express” on page 167
- “Creating relationship definitions” on page 173
- “Defining identity relationships” on page 174
- “Defining lookup relationships” on page 176
- “Creating the relationship table schema” on page 178
- “Copying relationship and participant definitions” on page 178
- “Renaming relationship or participant definitions” on page 179
- “Specifying advanced relationship settings” on page 179
- “Deleting a relationship definition” on page 183
- “Optimizing a relationship” on page 184

Overview of Relationship Designer Express

Relationship Designer Express is a graphical development tool for creating and modifying relationship definitions. A *relationship definition* establishes an association between two or more participants. You create a relationship definition by specifying the participants in the relationship and defining the data source and other properties associated with each participant.

This section provides the following information as an overview to Relationship Designer Express:

- “Starting Relationship Designer Express” on page 167
- “Working with projects” on page 168
- “Layout of Relationship Designer Express” on page 169
- “Customizing the main window” on page 170
- “Using the Relationship Designer Express functionality” on page 171

Starting Relationship Designer Express

To launch Relationship Designer Express, you can do any of the following:

- From System Manager, you can
  - Select Relationship Designer Express from the Tools menu.
  - Click a Relationship folder in a project to enable the Relationship Designer Express icon in the System Manager toolbar. Then click the Relationship Designer Express icon.
  - Right-click the Relationships folder in a project and select Relationship Designer Express from the Context menu.
  - Right-click a relationship in the Dynamic or Static folder and select Edit Definitions from the Context menu.

© Copyright IBM Corp. 2003

Result: Relationship Designer Express launches and highlights the selected relationship.

- From a development tool, such as Business Object Designer, Map Designer Express, or Process Designer, you can either:
  - Select Relationship Designer Express from the Tools menu.
  - Click the Relationship Designer Express icon in the Programs toolbar:
- Using a system shortcut:

```
Start-->Programs-->IBM WebSphere Business Integration Express
 for Item Sync v4.3 -->Toolset Express-->Development-->Relationship
 Designer Express
```

Important: For Relationship Designer Express to be able to access relationships stored in System Manager, Relationship Designer Express must be connected to an instance of System Manager. The preceding steps assume that you have already started System Manager. If you have *not* started System Manager, see the *User Guide for WebSphere Business Integration Express for Item Synchronization* for more information. If System Manager is already running, Relationship Designer Express will automatically connect to it.

Working with projects

System Manager is the only tool that interacts with the server. It imports and exports entities (relationships, maps) between InterChange Server Express and System Manager projects. Various tools, such as Relationship Designer Express, connect to System Manager and view, edit, and modify these entities on a project basis.

A *project* is simply a logical grouping of entities for managing and deployment purposes. Once entities are deployed to the InterChange Server Express, the project they originated from no longer has any meaning.

System Manager allows you to create multiple projects. Before you can work on a relationship, you must select which project the relationship is in.

To select a project to work with, perform the following steps:

1. Select Switch to Project from the File menu.
2. Select the name of the project in the Switch to Project submenu.

Result: You can now work with the relationships in that project. Before you can switch to yet another project, you are prompted to save the relationships you modified in the current project.

Figure 91 shows the Switch to Project option for browsing a project.

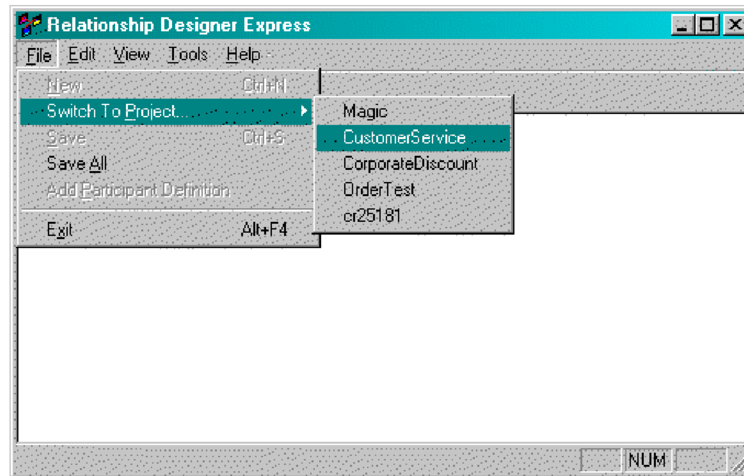


Figure 91. Browsing a project

When Relationship Designer Express establishes a connection to System Manager, it obtains a list of business objects that are defined in the current project. This list assists you with defining participants.

If you add or delete a business object using Business Object Designer, System Manager notifies Relationship Designer Express, which dynamically updates the list of business object definitions.

Layout of Relationship Designer Express

In the Relationship Designer Express window, a list of relationship definitions stored in the current project appears on the left side. In this relationship definition list, the contents of each relationship definition appear in a hierarchical format similar to the Windows Explorer. You can expand the relationship name by clicking on the plus symbol (+) beside its name to see a list of its participant definitions, participant types, and associated attributes. Figure 92 shows a relationship definition list.

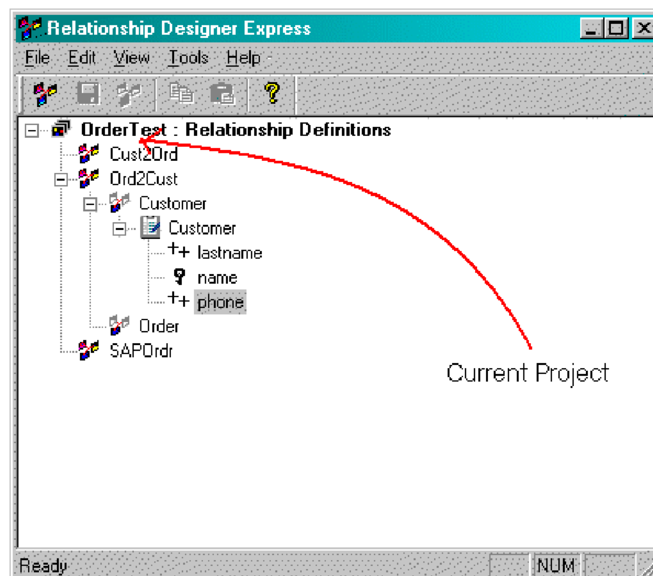


Figure 92. Relationship definition list

The Participant Types window shows a list of available data types in the current project that you can associate with a participant.

Figure 93 shows the main window of Relationship Designer Express, with both the Relationship Definition list and the Participant Types window.

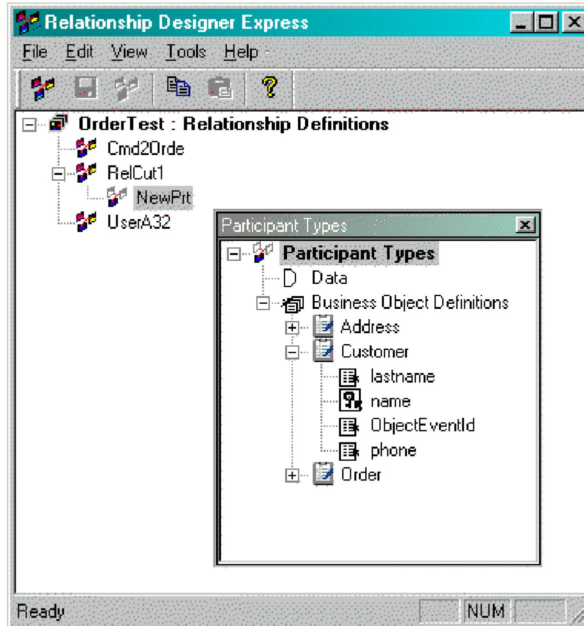


Figure 93. Relationship Designer Express main window

Customizing the main window

Relationship Designer Express provides the following ways to customize its main window:

- “Choosing windows to display” on page 170
- “Floating a dockable window” on page 171

Choosing windows to display

When you first open Relationship Designer Express, only the relationship definition list displays in the main window. The Participant Types window does not display. You can customize the appearance of the main window with options from the View pull-down menu. Table 63 describes the options of the View menu and how they affect the appearance of the Relationship Designer Express main window.

Table 63. View menu options for main window customization

View menu option	Element displayed
Participant Types	The Participant Types window displays
Toolbar	The Standard toolbar, which provides the main functionality for Relationship Designer Express
Status Bar	A single-line pane in which Relationship Designer Express displays status information

When a menu option appears with a check mark to the left, the associated element displays. To turn off display of the element, select the associated menu option. The check mark disappears to indicate that the element does not currently display. Conversely, you can turn on display of an undisplayed element by choosing the associated menu option. In this case, the check mark appears beside the displaying element.

Floating a dockable window

Relationship Designer Express supports the following portions of the main window as dockable windows:

- Standard toolbar
- Participant Types window

By default, a dockable window is usually placed along the edge of the main window and moves as part of the main window. When you float a dockable window, you detach it from the main window, allowing it to function as an independent window. To float a dockable window, hold down the left mouse button, grab the border of the window and drag it onto the main window or desktop.

Using the Relationship Designer Express functionality

You can access Relationship Designer Express's functionality using any of the following:

- The pull-down menus at the top of the window
- The Context menu
- Keyboard shortcuts
- The icons in the toolbars

Main menus of Relationship Designer Express

Relationship Designer Express provides the following pull-down menus:

- File menu
- Edit menu
- View menu
- Tools menu
- Help menu

The following sections describe the options of each of these menus. Keyboard shortcuts are available for some of these options, as indicated.

Functions of the File menu: The File pull-down menu of Relationship Designer Express displays the options shown in Table 64. Except for the Switch to Project option, all File menu options affect objects in the current project.

Table 64. File menu options in Relationship Designer Express

File menu option	Description	For more information
New (Ctrl+N)	Creates a new relationship definition	"Creating relationship definitions" on page 173
Switch to Project... (Ctrl+S)	A submenu with a list of other projects	"Working with projects" on page 168
Save	Saves the current relationship definition to a file	"Creating relationship definitions" on page 173
Save All	Saves all open relationship definitions	N/A

Table 64. File menu options in Relationship Designer Express (continued)

File menu option	Description	For more information
Add Participant Definition	Adds a new participant definition to the current relationship definition	“Creating relationship definitions” on page 173

Functions of the Edit menu: The Edit pull-down menu of Relationship Designer Express displays the following options:

- Rename—renames the relationship definition
- Copy (Ctrl+C)—Copies the current relationship definition.
- Paste (Ctrl+V)—Pastes the copied relationship definition.
- Cut (Ctrl+X)—Deletes the current relationship definition.
- Advanced Settings...—Displays the Advanced Settings window.

Functions of the View menu: The View pull-down menu of Relationship Designer Express displays the following options:

- Participant Types—Displays the Participant Types window.
- Expand Tree—Displays the members of the current level of the relationship definition list (same as clicking on the plus symbol beside the name of the level).
- Collapse Tree—Condenses the current level of the relationship definition list so that its members do not display (same as clicking on the minus symbol beside the name of the level).
- Toolbar—When on, Relationship Designer Express displays the Standard toolbar.
- Status Bar—When on, Relationship Designer Express can display its single-line status message at the bottom of the main window.

For information on the View menu options that control display, see “Choosing windows to display” on page 170.

Tools menu functions: The Tools pull-down menu of Relationship Designer Express provides options to start each of the WebSphere business integration tools:

- Relationship Manager
- Map Designer Express
- Business Object Designer Express

Help Menu functions: Relationship Designer Express provides a standard Help menu with the following options:

- Help Topics (F1)
- Documentation
- About...

Note: A Context menu provides shortcuts to useful commands and is available by right-clicking. Its options change depending on where you click

Relationship Designer Express toolbar

Relationship Designer Express provides a Standard toolbar for common tasks you need to perform. This toolbar is dockable; that is, you can detach it from the palette of the main window and float it over the main window or the desktop. Figure 94 shows the Relationship Designer Express Standard toolbar.

Figure 94. Relationship Designer Express Standard toolbar

The following list provides the function of each Standard toolbar button, left to right:

- New Relation
- Save Relation
- New Participant
- Copy
- Paste
- Help

Creating relationship definitions

Perform the following steps to create a relationship definition:

1. Create a relationship name with one of the following:
   - Select New Relationship Definition from the File menu.
   - Use the keyboard shortcut `Ctrl+N`.
   - In the Standard toolbar, click the New Relation button.
2. Name the icon for the relationship definition.

   Rule: Relationship definition names can be up to 8 characters long, can contain only letters and numbers, and must begin with a letter.
3. Create a participant definition for each business object to be related.

   To do so, select the relationship definition name and perform one of the following actions:
   - Select Add Participant Definition from the File menu.
   - In the Standard toolbar, click the New Participant button.
4. For each participant definition, name the icon for the participant definition.

   Rule: Participant definition names can be up to 8 characters long, can contain only letters and numbers, and must begin with a letter.
5. Associate a data type with each participant by dragging the type from the Participant Types window onto the participant definition.

   Tip: To display the Participant Types window, select Participant Types from the View menu.
   - To associate a business object data type, drag the business object definition from the Participant Types window.

     The participants in an identity relationship use business object definitions as their participant type. For more information, see "Defining identity relationships" on page 174.
   - To associate a Java data type, drag the `Data` participant type from the Participant Types window.

     In the relationship definition, the `Data` participant type represents *all* data types other than business object types. The participants in a lookup relationship use `Data` as their participant type. For more information, see "Defining lookup relationships" on page 176.
6. For participant types that are business object definitions, add or change the attributes to associate with the participant.

   The attributes you select become the basis on which the business objects are related.

7. Save the relationship definition with one of the following:
   - Select Save Relationship Definition from the File menu.
   - Use the keyboard shortcut `Ctrl+S`.
   - In the Standard toolbar, click the Save Relation button.
8. Before executing a map that uses the relationship definition, perform the following steps:
   - Activate the relationship. After the relationship is deployed to ICS Express, this new relationship is *not* activated. However, for the Mapping API methods to be able to access the relationship tables, a relationship table must be active. To activate the relationship, click the relationship name in System Manager and select the Start option from the Component menu. For more information about starting and stopping a relationship, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.
   - Compile and deploy the map that uses the relationship. If the map is deployed and compiled successfully in ICS Express, ICS Express creates the executable map code and activates the map. For more information, see “Compiling a map” on page 70.

Note: If you create or make a change to a relationship definition, you must first stop the relationship through the System Manager Relationship menu, make the change to the relationship, and then restart the relationship.

Defining identity relationships

An *identity relationship* establishes an association between two or more business objects on a one-to-one basis. That is, for a given relationship instance, there can be only one instance of each participant. You typically create an identity relationship to transform the key attributes in a business object, such as customer or product ID. For more background information, see “Identity relationships” on page 155.

InterChange Server Express supports the kinds of identity relationships shown in Table 65.

Table 65. Kinds of Identity relationships

Identity relationship type	Description	For more information
Simple identity relationship	Relates two business objects through a single key attribute	“Using simple identity relationships” on page 191
Composite identity relationship	Relates two business objects through a composite key (made up of more than one attribute)	“Using composite identity relationships” on page 202

To define an identity relationship using Relationship Designer Express, perform the following steps:

1. Create a relationship definition and the participant definitions by following Steps 1-4 in “Creating relationship definitions” on page 173.

   Guideline: Create a participant definition for each business object to be related. Identity relationships require participants for the generic business object as well as the application-specific business objects.
2. Associate a business object with each participant definition by dragging the business object definition from the Participant Types window onto the participant definition. You can release the drag button when the plus symbol (+) appears in the Relationship Designer Express main window. For

information on how to open the Participant Types window, see step 5 in "Creating relationship definitions" on page 173.

For identity relationships, the participant type is a business object. Every identity relationship has a participant with a participant type of the generic business object plus one participant for each application-specific business object.

3. For each business object that you associate with a participant definition, add the attributes that relate the business object with the other participants.

   To do so, expand the associated business object in the Participant Types window, select an attribute, and drag it onto the business object in the main Relationship Designer Express window. The attributes you select become the basis of the relationship between the business objects.

   For identity relationships, the attributes are usually the key attributes of each business object definition. The type of the key determines the kind of identity relationship:

   - For a single key, use a simple identity relationship. Each participant can consist of only one attribute: the unique key of the business object. For more information, see "Creating the child relationship definition" on page 201..
   - For a composite key, use a composite identity relationship. Specify a composite key by adding each key attribute in the order in which it appears in the composite key. Each participant can contain several attributes: usually, the unique key from the parent business object and at least one attribute from the child business object (within the parent business object). When deployed to the server, the relationship is saved in a table, the name of which is the concatenation of the attributes in the order in which they appear in the participant definition. For more information, including the index size limitations of some databases, see "Creating composite identity relationship definitions" on page 202..

4. Highlight the relationship definition name and select Advanced Settings from the Edit menu.

   Initially, the Advanced Settings window displays the relationship definition settings, as Figure 96 on page 180 shows.

   a. Modify the relationship definition settings as follows:
      - Under `Relationship type`, check the `Identity` box.

        Result: This setting tells InterChange Server Express to process the relationship as an identity relationship by setting a uniqueness constraint on the relationship instance ID and the key attributes for each participant. This action guarantees a one-to-one correspondence between all participants in each relationship instance.
      - If you want the relationship tables to reside in a database other than the default database (the WebSphere business integration system repository, by default), enter the appropriate database information in the `DBMS Settings` area of the window. For more information, see "Advanced settings for relationship definitions" on page 179.

   b. Modify the advanced settings for the participant definition.
      - In the object browser of the Advanced Settings window, expand the relationship definition and highlight the participant definition that represents the generic business object to display the participant definition settings (see Figure 97 on page 181). Check the box labeled `IBM WebSphere Business Integration Express for Item Sync- managed`.

        Result: This action tells Relationship Designer Express *not* to create relationship tables for the generic business object. When you maintain the

relationship with the `maintainSimpleIdentityRelationship()` method, the WebSphere business integration system uses the relationship instance IDs stored in the application-specific relationship tables to transform the relationship attributes.

- If you want to customize the name for this participant's relationship table or stored procedure, enter the name in the appropriate field in the window. For more information, see "Advanced settings for participant definitions" on page 180.

c. Click OK to close the Advanced Settings window.

5. Save the relationship definition as described in steps 7-8 in "Creating relationship definitions" on page 173.

Relating child business objects

When you create identity relationships, the business objects you are relating often have child business objects. For instance, some customer business objects have child business objects for storing address information. A child business object can participate in the kinds of relationships that Table 66 shows.

Table 66. Relationships for child business objects

Condition of child business object	Kind of relationship	For more information
The key for the child business object *uniquely* identifies the child beyond the context of its parent	Simple identity relationship	"Coding a child-level simple identity relationship" on page 201
The key for the child business object does *not* uniquely identify it beyond the context of its parent	Composite identity relationship	
To maintain the child business objects during an Update operation as part of the identity relationship	Parent/child relationship	"Managing child instances" on page 207

If the child is a multiple-cardinality child business object, you can change the index to make the participant reference a specific child. To do so, select the child's key attribute, right-click, and select Change Index from the Context menu. If the source and destination children in a map correspond one to one, the index is not significant and you do not need to change it. However, if the map transforms the children in any other way, you can enter a specific index number. For example, if the child business objects represent addresses and the third source address corresponds to the first destination address, you can change the indexes to `2` and `0`, respectively.

Defining lookup relationships

A *lookup relationship* associates data that is equivalent across business objects but may be represented in different ways. In this case, given a value in one business object, the relationship can look up its equivalent in the relationship tables for another business object. The most common example of attributes that might require lookups are codes (`EmployeeType`, `PayLevel`, `OrderStatus`) and abbreviations (`State`, `Country`, `Currency`). For more background information, see "Lookup relationships" on page 154.

When you create a relationship definition for a lookup, you add a participant definition for each business object that contains the attributes you want to relate.

However, you do not associate the actual business object definitions or attribute names with the participant definitions. Instead, you specify `Data` as the participant type for each participant definition.

To define a lookup relationship using Relationship Designer Express, perform the following steps:

1. Create a relationship definition and the participant definitions by following Steps 1-4 in "Creating relationship definitions" on page 173.

   Create a participant definition for each business object to be related.
2. For each participant definition, specify `Data` as the participant type by dragging the `Data` participant type from the Participant Types window onto the participant definition.

   In the relationship definition, the `Data` participant type represents *all* data types other than business object types. When you create the map and work with instances of the relationship using methods in the `Relationship`, `IdentityRelationship`, and `Participant` classes, you can use data of any of the supported Java data types, such as `String`, `int`, `long`, `float`, `double`, or `boolean`.
3. Make a note of the table name for storing the lookup values for each participant definition. You need to know the table name so you can populate the tables with the lookup values for each participant definition. Or, if you already have tables containing the lookup values, you can replace the generated table name with your own table name.

   To retrieve the table names for each participant definition in the relationship definition, or to specify your own table names:

   a. Select the participant definition and select Advanced Settings from the Edit menu.

      Result: The Advanced Setting dialog box appears showing the storage settings for that participant. See "Specifying advanced relationship settings" on page 179 for more information on these settings.
   b. Write down the storage settings for the participant, or overwrite the settings with your own table information.

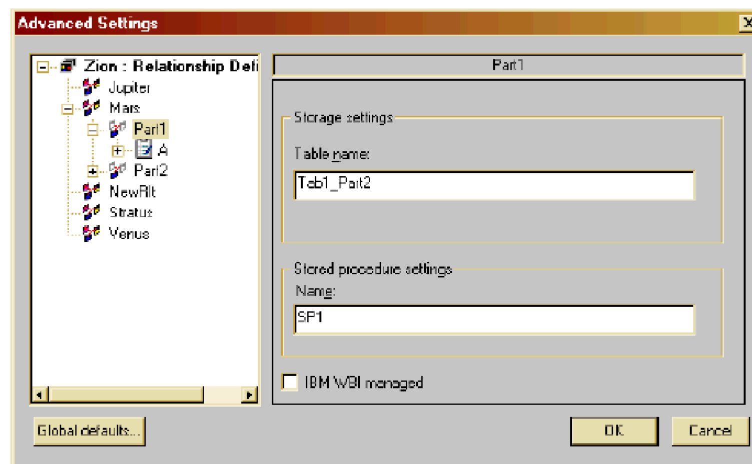


Figure 95. Advanced Settings dialog

   c. Repeat step 3a and step 3b for each participant definition.
   d. Click OK to close the Advanced Settings dialog box.

4. Save the relationship definition as described in steps 7-8 in “Creating relationship definitions” on page 173.

   To create the relationship tables, check the Create Schema box in the Deploy Project dialog in System Manager. For more information about when to create the run-time schema, see“Creating the relationship table schema” on page 178.
5. Using the information you gathered in step 3, populate the relationship tables with the lookup values for each participant, or add your own tables of lookup values to the database. For more information, see “Populating lookup tables with data” on page 189.

Creating the relationship table schema

For each relationship definition you create, InterChange Server Express uses the following database objects to maintain the run-time data for instances of the relationship:

- Tables in the relationship database hold the data of the relationship instances.
- Stored procedures in the relationship database maintain the relationship tables.

For information, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Copying relationship and participant definitions

Relationship Designer Express allows you to copy the following:

- Relationship definitions
- Participant definitions

Copying relationship definitions in the current project

To create a new relationship definition that is similar to an existing one, you can copy the existing definition and modify it to suit your needs. You can also copy a participant definition from a relationship definition and paste it into the same relationship definition or into another one.

To copy a relationship definition, perform the following steps:

1. Select the relationship definition you want to copy (for example, CustToClient) and select Save Relationship Definition from the File menu.
2. Select the relationship definition you want to copy and select Copy from the Edit menu.
3. Select the Project name (root tree node) and select Paste from the Edit menu.

   Result: Relationship Designer Express creates a new relationship definition with a name of `Copy of CustToClient`. The definition name appears in edit mode.
4. Enter a new name for the relationship definition, then press Enter.
5. To save the new definition to the repository, select Save Relationship Definition from the File menu (or use the keyboard shortcut `Ctrl+S`).

Tip: To copy a relationship definition from one InterChange Server Express to another, use the `repos_copy` command. The `repos_copy` command copies objects into and out of the InterChange Server Express repository. For more information on using `repos_copy`, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Copying participant definitions in the current project

To copy a participant definition:

1. Select the relationship definition to which the participant definition you want to copy belongs and select Save Relationship Definition from the File menu.
2. Select the participant definition you wish to copy and select Copy from the Edit menu.
3. Select the relationship definition to which you want to copy the participant definition and select Paste from the Edit menu.

   Result: Relationship Designer Express creates a new participant definition with a name of `Copy`. The definition name appears in edit mode.
4. Enter a new name for the participant definition, and then press Enter.

Renaming relationship or participant definitions

You can rename a relationship or participant definition before you save it to the repository. To change a definition's name after you have saved it, you must copy the definition to a new name and delete the old name. For help copying definitions, see "Copying relationship and participant definitions" on page 178.

Specifying advanced relationship settings

For each relationship definition you create, Relationship Designer Express maintains advanced settings that affect the storage and processing of the relationship instance data.

Note: If you change any database-related setting, such as a login account name, password, or a table name after creating the relationship table schemas, you must re-create the relationship table schemas using System Manager for your changes to take effect.

To view or change the settings, select Advanced Settings from the Edit menu. In the Advanced Settings dialog, the settings that appear on the right side differ depending on which of the following items you have selected on the left:

- Relationship definition
- Participant definition
- Attribute

Advanced settings for relationship definitions

To view or change the settings for a relationship definition, select the relationship name. The following illustration shows an example of the advanced settings at this

level:

Select the relationship definition name to view or change its settings.

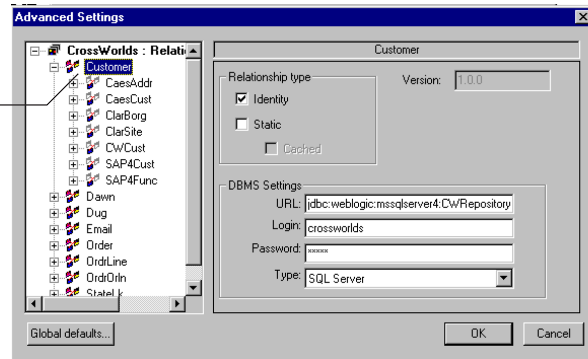


Figure 96. Advanced settings for a relationship definition

Table 67 summarizes the settings available for relationship definitions. Default values for the DBMS settings come from the Global Default Settings dialog box described in "Global default settings" on page 182:

Table 67. Summary of advanced settings for relationship definitions

Setting	Description
Relationship type	
Identity	When this option is enabled, relationship is an identity relationship. For more information, see "Defining identity relationships" on page 174.
Static	When this option is enabled, relationship is a static relationship. For more information, see "Defining lookup relationships" on page 176.
Cached	When the Static field is enabled, this field is enabled. Check this field to have the relationship tables cached in memory. For more information, see "Optimizing a relationship" on page 184.
Version	This field is read-only. Versions for relationship definitions are not supported in this release.
DBMS Settings	
URL	The JDBC path where the relationship tables for this relationship definition are located. The default location for all relationship tables is specified in Global Default Settings (see 182).
Login	The user name for logging in to the relationship database.
Password	The password for logging in to the relationship database.
Type	The relationship database type, such as SQL Server or DB2.

Note: If you specify a database for the relationship tables that is different from the InterChange Server Express's repository database, you might need to increase the setting for the maximum number of connection pools that the server can create. The server configuration parameter that specifies the number of connection pools is `MAX_CONNECTION_POOLS`. The default value is 10.

Advanced settings for participant definitions

To view or change the settings for participant definitions, select the participant definition name. The following illustration shows an example of the advanced

settings at this level:

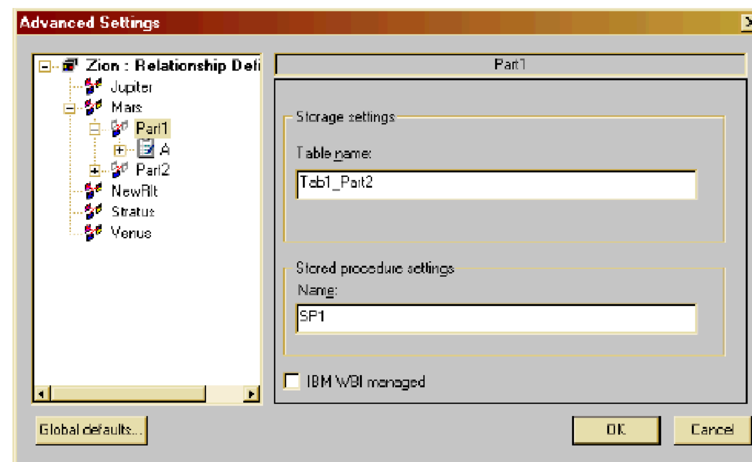


Figure 97. Advanced settings for a participant definition

Table 68 summarizes the settings available for participant definitions.

Table 68. Summary of advanced settings for participant definitions

Setting	Description
Table name	Name of the relationship table in the relationship database containing the relationship data for this participant instance. **Rule:** If your relationship database is a DB2 database, you *must* use up to a maximum of 18 characters in the relationship table names. Although table names do *not* have a limit in DB2, index names do. Because Relationship Designer Express generates index names for the relationship tables based on their table names, relationship table names for a DB2 database must be 18 characters or less.
Stored procedure name	Name of the stored procedure that maintains the relationship table.
IBM WBI managed	If checked, prevents relationship tables from being created for this participant. Check this setting only when: • The business object associated with this participant definition is a generic business object. • There is only one attribute associated with the participant and it is a key attribute.

Advanced settings for attributes

To view or change the advanced settings for an attribute, select the attribute. The following illustration shows an example of the advanced settings:

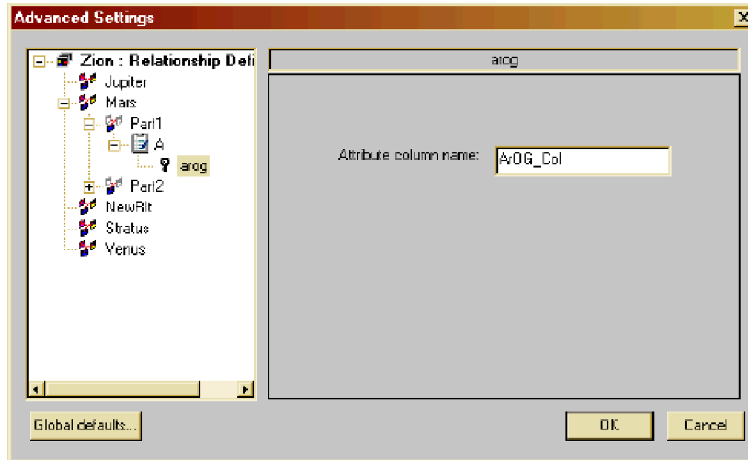


Figure 98. Advanced settings for attributes

For attributes, the only setting available is the attribute column name. The column name is the name of the column in the relationship table that contains the values for the selected attribute. It is typically the same as the attribute name. You might want to change the column name if you are using tables you created instead of the default tables that the Relationship Designer Express creates.

Global default settings

When you save a new relationship definition and create the relationship table schemas, Relationship Designer Express must know the location of the database for the relationship tables, the type of database, and how to access the database with a valid user name and password. Relationship Designer Express maintains default values for these settings, which it uses for all new relationship definitions you create. Once a relationship definition is created, these settings are stored with the relationship definition, and you can change the settings for each relationship definition individually.

By default, the database name and access information is the same one used by the InterChange Server Express repository. If you want to store your relationship tables in another location, you can modify the global settings.

To view or change the global default settings, perform the following steps:

1. In Relationship Designer Express, select Advanced Settings from the Edit menu.

   Result: The Advanced Settings dialog box appears.
2. Click the Global defaults button.

Result: The Global Default Settings dialog box appears.

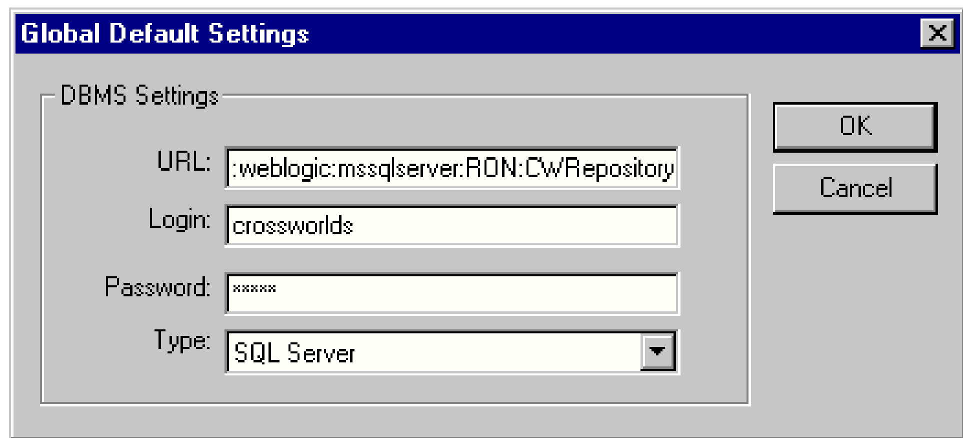


Figure 99. Global Default Settings dialog

Table 69 describes the global default settings for relationships.

Table 69. Relationship global default settings

Setting	Description
URL	The JDBC path where the relationship database is located. The default is the InterChange Server Express's repository database.
Login	The user name for logging in to the relationship database.
Password	The password for logging in to the relationship database.
Type	The relationship database type, such as SQL Server or DB2.

Note: If you specify a database for the relationship tables that is different from the InterChange Server Express's repository database, you might need to increase the setting for the maximum number of connection pools that the server can create. The server configuration parameter that specifies the number of connection pools is `MAX_CONNECTION_POOLS`. The default value is 10.

3. When you are finished viewing or making changes, click OK to save or Cancel to exit without saving.

Note: Changes that you make to the global default settings apply only to new relationship definitions. They do not affect existing relationships. If you want to change the settings for an existing relationship, see "Specifying advanced relationship settings" on page 179.

Deleting a relationship definition

The Relationship Designer Express allows you to delete a relationship definition listed in its main window by highlighting the definition and choosing Delete from the Edit menu or right-clicking on the definition and choosing Delete.

Optimizing a relationship

By default, each relationship's relationship tables are stored in the relationship database. Each time a relationship retrieves or modifies run-time data, it uses SQL statements to access this database. If the relationship tables are accessed frequently, these accesses can have a significant impact on performance in terms of CPU usage and InterChange Server Express resources. As part of the design of a relationship, you can determine whether to cache these relationship tables into memory.

To make this decision, you need to determine how frequently the relationship's run-time data changes. The WebSphere business integration system allows you to categorize your relationship in one of two ways:

- Dynamic relationship—a relationship whose run-time data changes frequently; that is, its relationship tables have frequent Insert, Update, or Delete operations. All relationships are dynamic by default.
- Static relationship—a relationship whose run-time data undergoes very minimal change; that is, its relationship tables have very few Insert, Update, or Delete operations. For example, because lookup tables store information such as codes and status values, their data very often is static. Such tables make good candidates for being cached in memory.

Note: The WebSphere business integration System Manager categorizes relationships into these same two categories. When you expand the Relationships folder, System Manager displays two subfolders: Dynamic and Static.

You define whether a relationship is dynamic or static from the Advanced Setting dialog for the relationship definition. The following sections summarize how to define a dynamic and static relationship from this dialog. For information on how to display the Advanced Setting dialog, see "Specifying advanced relationship settings" on page 179.

Defining a dynamic relationship

For a dynamic relationship, InterChange Server Express accesses the run-time data from its relationship tables in the relationship database. By default, InterChange Server Express assumes a relationship is dynamic. Therefore, you do not have to take any special steps to define a dynamic relationship:

- For an identity relationship, click Identity from the Advanced Settings dialog, as described in "Defining identity relationships" on page 174.
- For a lookup relationship, make sure Identity is *not* checked, as described in "Defining lookup relationships" on page 176.

Note: For a dynamic relationship, do *not* click the Static or Cached field on the Advanced Settings dialog.

System Manager lists all dynamic relationships in the folder labeled Dynamic under the Relationships folder.

Defining a static relationship

For a static relationship, InterChange Server Express can access the run-time data from cached relationship tables. With caching enabled for the static relationship, InterChange Server Express stores a copy of the relationship tables in memory. When making the decision to cache relationship tables, try to balance the following conditions:

- Performance usually improves if you let InterChange Server Express cache the relationship tables in memory.

  In this case, the server does not need to use SQL statements to access the relationship database for the run-time data. Instead, it can access memory for this data, which is much faster. If the run-time data for a static relationship is not currently in memory, InterChange Server Express reads the appropriate relationship tables from the database into memory when the data is first accessed. For future accesses, InterChange Server Express uses the cached version of the tables.

  However, once the table is read into memory, InterChange Server Express must maintain consistency between the relationship tables in the database and the cached tables. For Update, Insert, and Delete operations, InterChange Server Express must modify *both* the database tables *and* the cached tables. This double update can be very performance intensive. When you determine whether to cache a relationship's tables, consider the expected lifetime and refresh rate of the data.
- Memory usage increases when relationship tables are cached in memory. The amount of memory used is roughly equivalent to the size of all in-memory tables.

  Recommendation: You should not cache a relationship table that contains more than 1000 rows.

  Important: InterChange Server Express does *not* check for excessive memory usage. You must ensure that memory usage remains within the limits that your system imposes.

To define a static relationship, display the Advanced Settings dialog (see Figure 96) for the relationship definition and set the Static field from this dialog as follows:
- For an identity relationship, enable both the Identity and Static fields. For more information on the use of the Identity field, see "Defining identity relationships" on page 174.
- For a lookup relationship, enable the Static field (*not* the Identity field).

When the Static field is enabled, the Advanced Settings dialog also enables the Cached field. The Cached field allows you to control when InterChange server actually caches the relationship's table:
- When Cached is enabled, InterChange server can cache the relationship tables for a static relationship. It caches *all* relationship tables involved in the relationship.
- When Cached is disabled, InterChange Server Express does not cache the relationship tables in memory. Instead, it uses the tables in the relationship database for future accesses.

You can only control caching for a relationship that is defined as static.

Note: After you change a relationship's static or cached state from the Advanced Settings dialog, make sure you save the relationship definition for the change to be stored in the project.

Note: You can modify the cached and reload relationship properties from the server component management view. To do this, right-click the static relationship and select the properties from the Context menu.
- Cached—controls caching of the relationship's tables.

- Reload—tells InterChange Server Express to reread the relationship’s tables into memory.

For more information on caching and reloading relationship tables, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Chapter 8. Implementing relationships

Relationship attributes are those you transform using relationships. You do *not* transform relationship attributes by dragging from source attribute to destination attribute. Instead, you create a Custom transformation and customize the transformation rule for the destination relationship attribute using the function blocks in Activity Editor.

This chapter describes how to develop code within a map to implement the different kinds of relationships.

Note: This chapter assumes that you have already created the relationship definitions for the relationships. For information, see Chapter 7, “Creating relationship definitions,” on page 167.

- “Implementing a relationship” on page 187
- “Using lookup relationships” on page 188
- “Using simple identity relationships” on page 191
- “Using composite identity relationships” on page 202
- “Managing child instances” on page 207
- “Setting the verb” on page 210
- “Performing foreign key lookups” on page 216
- “Loading and unloading relationships” on page 221

Implementing a relationship

Once you have created a relationship definition within Relationship Designer Express, you are ready to implement the relationship within the map.

Note: See Chapter 7, “Creating relationship definitions,” on page 167 for instructions about how to create relationship definitions.

To implement a relationship, you use the relationship function blocks in the map’s destination object. Table 70 shows the function blocks to use.

Table 70. Relationship function blocks

Kind of relationship	Function block	For more information
Lookup	General/APIs/Relationship/Retrieve Instances General/APIs/Relationship/Retrieve Participants	“Using lookup relationships” on page 188
Simple identity	General/APIs/Identity Relationship/Maintain Simple Identity Relationship General/APIs/Identity Relationship/Maintain Child Verb	“Using simple identity relationships” on page 191
Composite Identity	General/APIs/Identity Relationship/Maintain Composite Relationship General/APIs/Identity Relationship/Maintain Child Verb General/APIs/Identity Relationship/Update My Children (optional)	“Using composite identity relationships” on page 202

© Copyright IBM Corp. 2003

Table 70. Relationship function blocks (continued)

Kind of relationship	Function block	For more information
Custom	General/APIs/Relationship/Create Relationship General/APIs/Identity Relationship/Add My Children General/APIs/Relationship/Add Participant	

When transforming relationship attributes, a map needs to know the calling context of the map. To determine calling context, the map needs the following information from the map execution context:

- The map's calling context, which is part of the map execution context

  For more information, see "Calling contexts" on page 146..
- The verb, which is part of the business object

These two factors tell the map what actions need to be taken on the relationship tables.

Using lookup relationships

A *lookup relationship* associates data that is equivalent across business objects but may be represented in different ways. The following sections describe the steps for using lookup relationships:

- "Creating lookup relationship definitions"
- "Populating lookup tables with data" on page 189
- "Customizing map transformations for a lookup relationship" on page 191

Note: For background information, see "Lookup relationships" on page 154..

Creating lookup relationship definitions

Lookup relationship definitions differ from identity relationship definitions in that the participant types are *not* business objects but of the type `Data` (the first selection in the participant types list). For more information on how to create a relationship definition for a lookup relationship, see "Defining lookup relationships" on page 176.

For example, suppose you create a lookup relationship called `StatAdtp` for the `AddressType` values. In Figure 100, each box represents a participant in the `StatAdtp` lookup relationship. Notice that each participant in this relationship is of type `Data`.

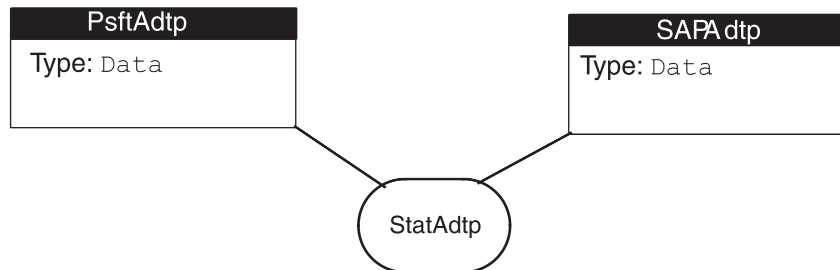


Figure 100. The StatAdtp lookup relationship definition

Because a lookup relationship does not indicate which attributes are being related, you can use one lookup relationship definition for transforming several attributes. In fact, you can use one lookup relationship definition for every attribute that requires a lookup, regardless of the business object being transformed. However, because only one set of tables is created for each relationship definition, using one relationship definition for all lookup relationships would make the tables large and hard to maintain.

A better strategy might be to create one lookup relationship definition per common unit of data, such as country code or status. This way, each set of relationship tables contains information related by meaning. Relationships defined this way are also more modular because you can add new participants, as you support new collaborations or applications, and reuse the same relationship definition. For instance, suppose you create a lookup relationship definition for country code to transform `Clarify_Site` business objects to `SAP_Customer`. Later on, if you add new collaborations or a new application, you can reuse the same relationship definition for every transformation involving a country code.

Populating lookup tables with data

When you deploy the lookup relationship definition with the option Create Schema enabled, Interchange Server Express generates a relationship table (also called a *lookup table*) for each participant. Each lookup table has a name of the form:

`RelationshipDefName_ParticipantDefName`

When you save the `StatAdtp` relationship definition (see Figure 100) with the option Create Schema enabled, Interchange Server Express generates the following two lookup tables:

- `StatAdtp_PsftAdtp_T`
- `StatAdtp_SAPAdtp_T`

A lookup table contains a column for the relationship instance ID (`INSTANCEID`) and its associated participant instance data (`data`). Figure 101 shows the lookup tables for the `PsftAdtp` and `SAPAdtp` participants in the `StatAdtp` lookup relationship. These two lookup tables use the relationship instance ID to correlate the participants. For example, the instance ID of 116 correlates the `PsftAdtp` value of `Fired` and the `SAPAdtp` value of `04`.

StatAdtp_PsftAdtp_T

PsftAdtp	
INSTANCEID	data
114	Active
115	Inactive
116	Fired
117	Retired

StatAdtp_SAPAdtp_T

SAPAdtp	
INSTANCEID	data
115	03
116	04
117	05
118	02

Figure 101. Relationship tables for the CustLkUp lookup relationship

Unlike relationship tables that hold data for identity relationships, lookup tables do *not* get populated automatically. You must populate these tables by inserting data into their columns. You can populate a lookup table in either of the following ways:

- Create a script that contains SQL INSERT statements to fill the lookup table with the desired data.
- Use Relationship Manager to add rows to the lookup table.

Inserting participant instances with SQL

You can insert participant data into a lookup table with the SQL statement INSERT. This method is useful when you need to add many rows of data to the lookup table. You can create the syntax for one INSERT statement and then use the editor to copy and paste this line as many times as you have rows to insert. In each line, you only have to edit the data to be inserted (usually in a VALUES clause of the INSERT statement).

To use the INSERT statement, you must know the name of the lookup relationship table and its columns. Table 71 shows the column names in a lookup table.

Table 71. Columns of a lookup table

Column in lookup table	Description
INSTANCEID	The relationship instance ID.
data	The participant data
STATUS	Set to zero (0) when the participant is active
LOGICAL_STATE	Indicates whether the participant instance has been logically deleted (zero indicates "no")
TSTAMP	Date of last modification for the participant instance.

Attention: When you use SQL statements to insert participant data into a lookup table, make sure you provide a value for the STATUS, LOGICAL_STATE, and TSTAMP columns. All values are required for IBM WebSphere business integration tools to function correctly. In particular, omission of the TSTAMP value causes Relationship Manager to be unable to retrieve the participant data; if no timestamp value exists, Relationship Manager raises an exception.

Suppose you want to add the participant data in to the relationship table that contains information for address type, shown in Table 72.

Table 72. Sample values for address type for PsftAdtp participant

INSTANCEID	STATUS	LOGICAL_STATE	TSTAMP	data
1	0	0	current date	Home
2	0	0	current date	Mailing

The following INSERT statements create the Table 72 participant data in the PstfAdtp lookup table:

```
INSERT INTO StatAdtp_PsftAdtp_T

   (INSTANCEID, STATUS, LOGICAL_STATE, TSTAMP, data)
   VALUES (1, 0, 0, getDate(), 'Home')

INSERT INTO StatAdtp_PsftAdtp_T
   (INSTANCEID, STATUS, LOGICAL_STATE, TSTAMP, data)
   VALUES (2, 0, 0, getDate(), 'Mailing')
```

Note: The preceding `INSERT` syntax is compatible with the MicroSoft SQL Server 7.0. If you are using another database server for your relationship table, make sure you use `INSERT` syntax compatible with that server.

Inserting participant instances with Relationship Manager

Relationship Manager is an IBM WebSphere business integration tool that graphically displays run-time data in a relationship table. Relationship Manager is useful when you only need to add a few rows to the lookup table. For more information on Relationship Manager, see the *User Guide for WebSphere Business Integration Express for Item Synchronization*.

Customizing map transformations for a lookup relationship

Once you have created the relationship definition and participant definitions for the lookup relationship, you can customize the map transformation rule for performing the lookups. For information on using lookup relationships, see "Example 3 of using the Activity Editor" on page 133.

Using simple identity relationships

An identity relationship establishes an association between business objects or other data on a *one-to-one basis*. A simple identity relationship relates two business objects through a single key attribute. The following sections describe the steps for working with simple identity relationships:

- "Creating simple identity relationship definitions"
- "Accessing identity relationship tables"
- "Defining transformation rules for a simple identity relationship" on page 201

Creating simple identity relationship definitions

Identity relationship definitions differ from lookup relationship definitions in that the participant types are business objects, *not* of the type `Data` (the first selection in the participant types list). For a simple identity relationship, the relationship consists of the generic business object and at least one application-specific business object. The participant type for a simple identity relationship is a business object for *all* participants. The participant attribute for every participant is a single key attribute of the business object. (For more information on how to create a relationship definition for a simple identity relationship, see "Defining identity relationships" on page 174.)

Accessing identity relationship tables

To reference a simple identity relationship, define a Cross-Reference transformation rule between the application-specific business object and the generic business object. For more information, see "Cross-referencing identity relationships" on page 45.

For example, the `CustIden` relationship (see Figure 82) transforms a `SiteID` key attribute in a Clarify customer to an `RefID` key attribute in an SAP customer. It includes maps between the following objects:

- Inbound map: `Clarify_Site` to `Customer`

  Obtain from the `ClarCust` relationship table the relationship instance ID that is associated with the `SiteID` key value.
- Outbound map: `Customer` to `SAP_Customer`

  Obtain from the `SAPCust` relationship table the `RefID` key value that is associated with relationship instance ID.

Figure 102 shows how to use the `CustIden` relationship tables to transform a `SiteID` value of `A100` to a `RefID` value of `806`.

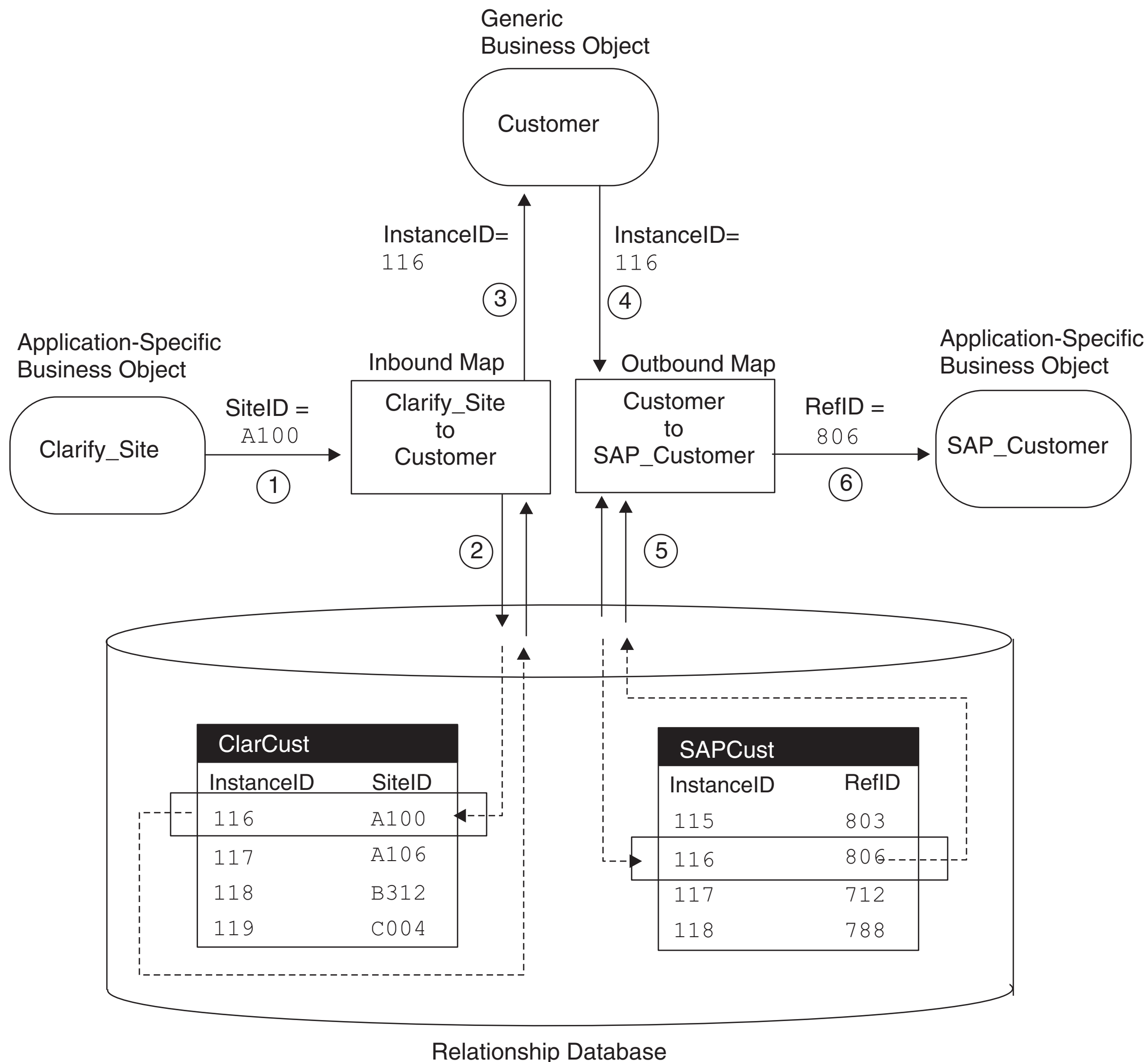


Figure 102. Using relationship tables to transform a SiteID to a RefID

The maintainSimpleIdentityRelationship() method must manage the relationship tables to ensure that related application-specific keys remain associated to a single relationship instance ID. At a high level, the Cross-Reference transformation rule generates code to do the following:

1. Perform validations on the arguments that are passed in. If an argument is invalid, the method throws the `RelationshipRuntimeException` exception. For a list of validations that the Java code generated by the Cross-Reference transformation performs, see the `maintainSimpleIdentityRelationship()` API in Chapter 20, "IdentityRelationship class," on page 345.
2. Takes the appropriate actions to maintain the relationship tables based on the calling context, which includes the following factors:
   - The verb of the business object

     The Cross-Reference transformation obtains this verb from the source business object. For inbound maps, the source is the application-specific business object; for outbound maps, the source is the generic business object.

- The value of the calling context

  The Cross-Reference transformation rule obtains the calling context from the map execution context automatically.

This transformation deals with the calling contexts shown in Table 73..

Table 73. Calling contexts with maintainSimpleIdentityRelationship()

Calling context	Description
EVENT_DELIVERY	A connector has sent an event from the application to InterChange Server Express (event-triggered flow).
ACCESS_REQUEST	An access client has sent an access request from an external application to InterChange Server Express.
SERVICE_CALL_REQUEST	A collaboration is sending a business object down to the application through a service call request.
SERVICE_CALL_RESPONSE	A business object was received from the application as a result of a successful response to a collaboration service call request.
SERVICE_CALL_FAILURE	A collaboration's service call request has failed. As such, corrective action might need to be performed.
ACCESS_RESPONSE	The source business object is sent back to the source access client in response to a subscription delivery request.

The following sections discuss the behavior of the Cross-Reference transformation with each of the calling contexts in Table 73..

EVENT_DELIVERY and ACCESS_REQUEST calling contexts

When the calling context is EVENT_DELIVERY or ACCESS_REQUEST, the map is being called is an inbound map; that is, it transforms an application-specific business object to a generic business object. A connector sends the EVENT_DELIVERY calling context; an access client sends an ACCESS_REQUEST calling context. In either case, the inbound map receives an application-specific business object as input and returns a generic business object as output. Therefore, the task for the Cross-Reference transformation is to obtain from the relationship table a relationship instance ID for a given application-specific key value.

For the EVENT_DELIVERY and ACCESS_REQUEST calling contexts, the Java code generated by the Cross-Reference transformation takes the following actions:

1. Locate the relationship instance in the relationship table that matches the given application-specific business object's key value. Table 74 shows the actions that the Java code generated by the Cross-Reference transformation takes on the relationship table based on the verb of the application-specific business object.
2. Obtain the instance ID from the retrieved relationship instance.
3. Copy the instance ID into the generic business object.

Table 74. Actions for the EVENT_DELIVERY and ACCESS_REQUEST Calling Contexts

Verb of application-specific business object	Action Performed by maintainSimpleIdentityRelationship()
Create	Insert a new entry into the relationship table for the application-specific business object's key value. If an entry for this key value already exists, retrieve the existing one; do *not* add another one to the table.

Table 74. Actions for the EVENT_DELIVERY and ACCESS_REQUEST Calling Contexts (continued)

Verb of application-specific business object	Action Performed by maintainSimpleIdentityRelationship()
Update	Retrieve the relationship entry from the relationship table for the given application-specific business object's key value. If an entry for this key value does *not* exist, add one to the table.
Delete	1. Retrieve the relationship entry from the relationship table for the given application-specific business object's key value. 2. Mark the relationship entry as "deactive".
Retrieve	Retrieve the relationship entry from the relationship table for the given application-specific business object's key value. If an entry for this key value does *not* exist, throw a RelationshipRuntimeException exception.

For an identity relationship that supports the transformation of an AppA application-specific business object to AppB application-specific business object, Figure 103 shows how the Java code generated by the Cross-Reference transformation accesses a relationship table associated with the AppA participant when a calling context is EVENT_DELIVERY (or ACCESS_REQUEST) and the AppA application-specific business object's verb is either Create or Update.

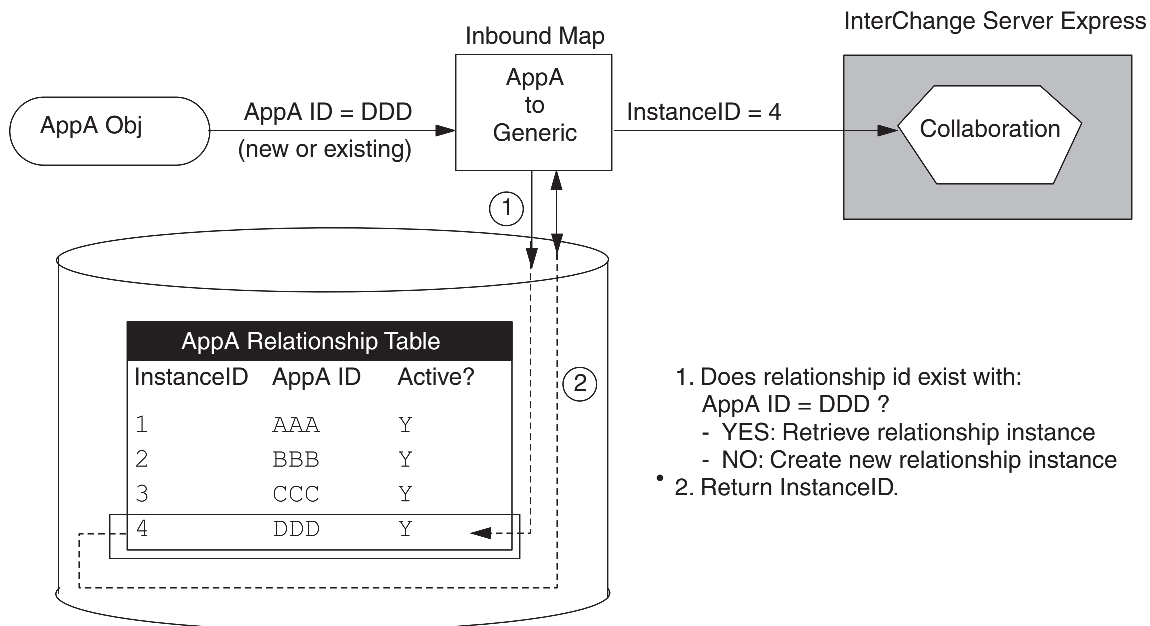


Figure 103. EVENT_DELIVERY and ACCESS_REQUEST with a create or update verb

For a calling context of EVENT_DELIVERY (or ACCESS_REQUEST) and an application-specific verb of either a Create or Update, Figure 104 shows the write that the Java code generated by the Cross-Reference transformation makes to the relationship table when no entry exists that matches the AppA application-specific key value.

Before Create

AppA Relationship Table		
InstanceID	AppA ID	Active?
1	AAA	Y
2	BBB	Y
3	CCC	Y

After Create

AppA Relationship Table		
InstanceID	AppA ID	Active?
1	AAA	Y
2	BBB	Y
3	CCC	Y
4	DDD	Y

New Relationship Entry

Figure 104. The Write to the relationship table for a new relationship entry

For a calling context of EVENT_DELIVERY (or ACCESS_REQUEST) and an application-specific verb of Delete, Figure 105 shows the write that the Java code generated by the Cross-Reference transformation performs on the AppA relationship table.

Before Delete

AppA Relationship Table		
InstanceID	AppA ID	Active?
1	AAA	Y
2	BBB	Y
3	CCCDDD	Y
4		Y

After Delete

AppA Relationship Table		
InstanceID	AppA ID	Active?
~~1~~	~~AAA~~	~~Y~~
2	BBB	N
3	CCC	Y
4	DDD	Y

“Deleted” Row

Figure 105. The write to the relationship table for a delete verb

SERVICE_CALL_REQUEST calling context

When the calling context is SERVICE_CALL_REQUEST, the map is being called is an outbound map; that is, it transforms a generic business object to an application-specific business object. The outbound map receives a generic business object as input and returns an application-specific business object as output. Therefore, the task for the Cross-Reference transformation is to obtain from the relationship table an application-specific business object’s key value for a given a relationship instance ID *only* if the verb is Update, Delete, or Retrieve. The Cross-Reference transformation does *not* obtain the application-specific key value for a Create verb.

Table 75 shows the action that the Cross-Reference transformation takes on the relationship table based on the verb of the generic business object.

Table 75. Actions for the SERVICE_CALL_REQUEST calling context

Verb of generic business object	Action performed by the Cross-Reference transformation
Create	Take no action. When the calling context is SERVICE_CALL_RESPONSE, the method actually writes a new entry to the relationship table. For more information, see “SERVICE_CALL_RESPONSE calling context” on page 197.

Table 75. Actions for the SERVICE_CALL_REQUEST calling context (continued)

Verb of generic business object	Action performed by the Cross-Reference transformation
Update Delete Retrieve	1. Obtain the generic business object's key value (the relationship instance ID) from the original-request business object in the map execution context. 2. Retrieve the entry from the relationship table for the given generic business object's key value. If an entry for this key value does *not* exist, throw a `RelationshipRuntimeException` exception. If no participants are found when the verb is Retrieve, throw a `CxMissingIDException` exception. 3. Obtain the application-specific key value from the retrieved relationship entry. 4. Copy the application-specific key value into the application-specific business object.

As Table 75 shows, when the verb is Create, the Java code generated by the Cross-Reference transformation does *not* write a new entry to the relationship table. It does not perform this write operation because it does not yet have the application-specific key value that corresponds to the relationship instance ID. When the connector processes the application-specific business object, it notifies the application of the need to insert a new row (or rows). If this insert is successful, the application notifies the connector, which creates the appropriate application-specific business object with a Create verb and the application's key value.

For the remaining verbs (Update, Delete, and Retrieve), the Java code generated by the Cross-Reference transformation performs a read operation on the relationship table. For an identity relationship that supports the transformation of an `AppA` application-specific business object to AppB application-specific business object, as Figure 106 shows how the Cross-Reference transformation accesses a relationship table associated with the `AppB` participant when a calling context is `SERVICE_CALL_REQUEST` and the generic business object's verb is Update, Delete, or Retrieve.

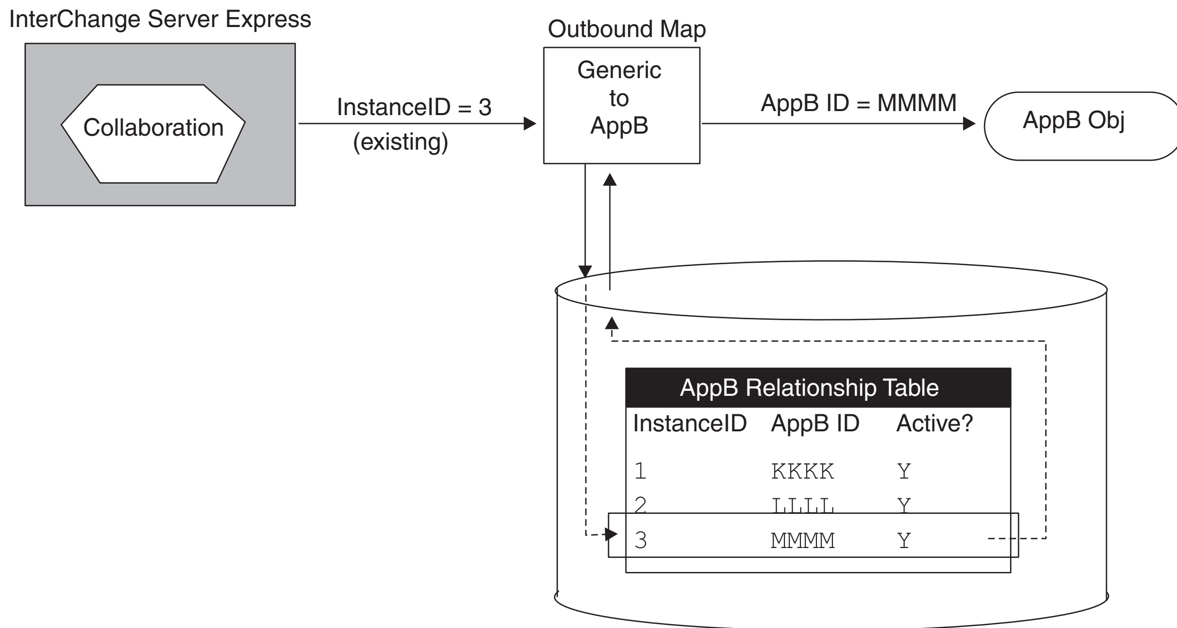


Figure 106. SERVICE_CALL_REQUEST with an update, delete, or retrieve verb

SERVICE_CALL_RESPONSE calling context

When the calling context is SERVICE_CALL_RESPONSE, the map is being called is an inbound map; that is, it transforms an application-specific business object to a generic business object. The inbound map receives an application-specific business object as input and returns a generic business object as output. The SERVICE_CALL_RESPONSE calling context is important for the Create verb, to indicate that the destination application was able to create a unique value for the new entity and the connector has returned an application-specific business object.

The task for the Cross-Reference transformation rule is to maintain an application-specific business object's key value in the relationship table for an existing relationship instance ID. For the SERVICE_CALL_RESPONSE calling context, the Java code generated by the Cross-Reference transformation takes the following actions:

1. Determines whether the generic business object is null:
   - For the Update, Delete, and Retrieve verbs, the transformation throws the RelationshipRuntimeException if the generic business object is null.
   - For a Create verb, a null-valued generic business object is valid.
2. Locates the entry in the relationship table that matches the given application-specific business object's key value. Table 76 shows the action that the Java code generated by the Cross-Reference transformation takes on the relationship table based on the verb of the application-specific business object.

Table 76. Actions for the SERVICE_CALL_RESPONSE calling context

Verb of application-specific business object	Action performed by maintainSimpleIdentityRelationship()
Create	For the given application-specific key, insert into the relationship table the new relationship entry containing the application-specific business object's key value and its associated relationship instance ID. The method obtains the relationship instance ID from the original-request business object in the map execution context (cwExecCtx). If an entry for this key value already exists, retrieve the existing one; do *not* add another one to the table.
Delete	1. Retrieve the relationship entry from the relationship table for the given application-specific business object's key value. 2. Mark the relationship entry as "deactive."
Update	Retrieve the relationship entry from the relationship table for the given application-specific business object's key value.
Retrieve	Retrieve the relationship entry from the relationship table for the given application-specific business object's key value.

For an identity relationship that supports the transformation of an AppA application-specific business object to AppB application-specific business object, Figure 107 shows how the Java code generated by the Cross-Reference transformation accesses a relationship table associated with the AppB participant when a calling context is SERVICE_CALL_RESPONSE and the AppB application-specific business object's verb is Create.

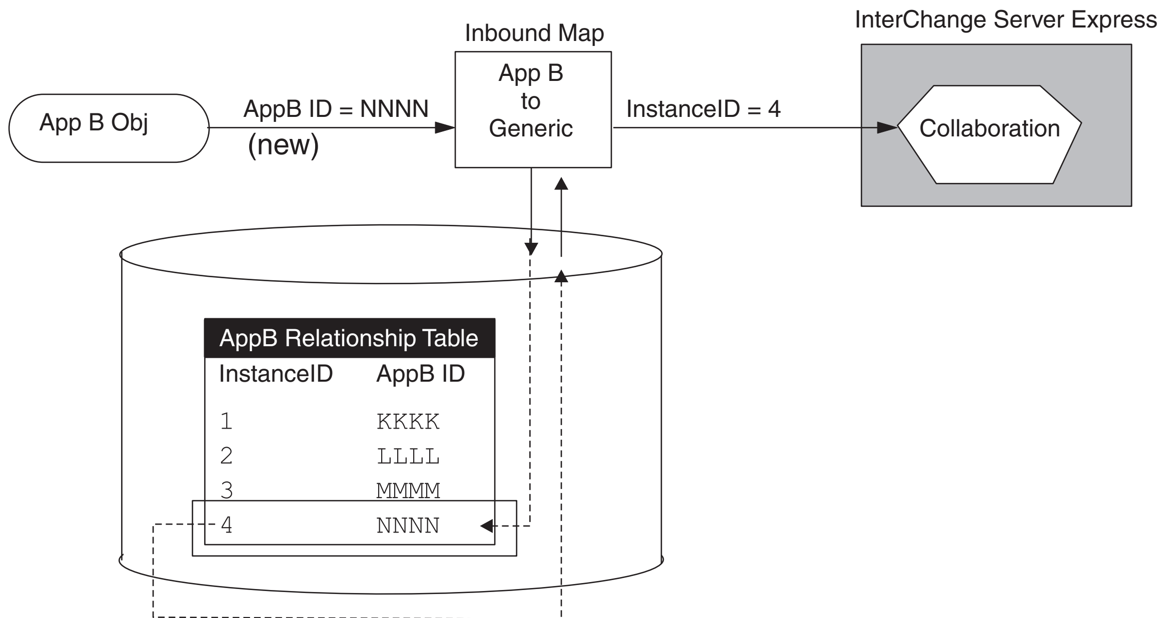


Figure 107. SERVICE_CALL_RESPONSE with the create verb

When the calling context is SERVICE_CALL_RESPONSE and the verb is Create, the inbound map has been invoked by the connector controller in response to the following actions:

- The connector has been notified that the application has inserted a new row.

  The connector sent this insert request to the application when it received the application-specific business object with a Create verb from the outbound map. This outbound map had a calling context of SERVICE_CALL_REQUEST. When the calling context was SERVICE_CALL_REQUEST, the Cross-Reference transformation could not write a new relationship instance to the relationship table because it did not yet have the application-specific key value that corresponded to the instance ID.
- The connector has generated a new application-specific business object based on the values in the new application-specific row and with a verb of Create.

  The connector sends this application-specific business object to InterChange Server Express, where it is received by the connector controller.
- The connector controller has called the inbound map to convert the application-specific business object to a generic business object.

  The inbound map contains a Cross-Reference transformation to create an entry in the relationship table for the new application-specific key.

For a calling context of SERVICE_CALL_RESPONSE and an application-specific verb of Create, Figure 108 shows the write that the Java code generated by the Cross-Reference transformation makes to the relationship table.

Before Create

AppB Relationship Table		
InstanceID	AppB ID	Active?
1	KKKK	Y
2	LLLL	Y
3	MMMM	Y

After Create

AppB Relationship Table		
InstanceID	AppB ID	Active?
1	KKKK	Y
2	LLLL	Y
3	MMMM	Y
4	NNNN	Y

"Inserted" Row

Figure 108. The write to the relationship table for a create verb

The Cross-Reference transformation must associate the new `AppB` application-specific key with its equivalent value in the `AppA` application. For the `EVENT_DELIVERY` or `ACCESS_REQUEST` calling context, the Cross-Reference transformation could just generate a new relationship instance ID. However, for `SERVICE_CALL_RESPONSE`, the Cross-Reference transformation cannot just generate a new instance ID. Instead, it must assign the same relationship instance ID to the `AppB` key value as it has already assigned to the `AppA` key value. The method obtains the instance ID associated with the `AppA` key value from the original-request business object, which is part of the map execution context.

In Figure 108, the Java code generated by the Cross-Reference transformation takes the following steps for the `SERVICE_CALL_RESPONSE` calling context and the Create verb:

- Obtain the instance ID of `4` from the original-request business object in map execution context.
- Create a new entry in the `AppB` relationship table for this instance ID (`4`) and the new application-specific key (`NNNN`).

When the map executions with both the `EVENT_DELIVERY` (or `ACCESS_REQUEST`) and `SERVICE_CALL_RESPONSE` calling contexts (and a Create verb) are complete, the relationship tables for `AppA` and `AppB` use common relationship instance IDs to associate their keys, as Figure 109 shows.

AppA Relationship Table		
InstanceID	AppA ID	Active?
1	AAA	Y
2	BBB	Y
3	CCC	Y
4	DDD	Y

AppB Relationship Table		
InstanceID	AppB ID	Active?
1	KKKK	Y
2	LLLL	Y
3	MMMM	Y
4	NNNN	Y

Relationship Instance

Figure 109. Creating the relationship instance

For the Update and Delete verbs (and Retrieve, if the instance ID already exists in the relationship table), the Cross-Reference transformation just retrieves the relationship instance ID from the relationship table. For a calling context of `SERVICE_CALL_RESPONSE` and an application-specific verb of Delete, the Cross-Reference transformation must take an additional step to deactivate the relationship instance, as Figure 110 shows.

Before Delete

AppB Relationship Table		
InstanceID	AppB ID	Active?
1	KKKK	Y
2	LLLL	Y
3	MMMM	Y
4	NNNN	Y

After Delete

AppB Relationship Table		
InstanceID	AppB ID	Active?
1	KKKK	Y
2	LLLL	N
3	MMMM	Y
4	NNNN	Y

"Deleted" Row

Figure 110. The write to the relationship table for SERVICE_CALL_RESPONSE and a delete verb

SERVICE_CALL_FAILURE calling context

When the calling context is `SERVICE_CALL_FAILURE`, the map is being called is an inbound map; that is it transforms an application-specific business object to a generic business object. For `SERVICE_CALL_FAILURE`, the inbound map receives an null application-specific business object as input and returns a generic business object as output. The `SERVICE_CALL_FAILURE` calling context is important for the Create verb; it indicates that the destination application was unable to create a unique value for the new entity and therefore the connector was unable to return an application-specific business object. The task for the Cross-Reference transformation is the same for all verbs, as Table 77 shows.

Table 77. Actions for the SERVICE_CALL_FAILURE calling context

Verb of Application-Specific business object	Action Performed by maintainSimpleIdentityRelationship()
Create Delete Update Retrieve	1. Obtain the key value (relationship instance ID) from the generic business object. This generic business object is in the map execution context. 2. Copy the retrieved instance ID into the generic business object.

ACCESS_RESPONSE calling context

When the calling context is `ACCESS_RESPONSE`, the map is being called is an outbound map as a result of a call-triggered flow. It transforms a generic business object to an application-specific business object. The outbound map receives a generic business object as input and returns an application-specific business object as output. Therefore, the task for the Cross-Reference transformation is the same for all verbs, as Table 78 shows.

Table 78. Actions for the ACCESS_RESPONSE calling context

Verb of generic business object	Action Performed by maintainSimpleIdentityRelationship()
Create Delete Update Retrieve	1. Obtain the key value (relationship instance ID) from the generic business object. This generic business object is in the map execution context. 2. Convert the relationship instance ID to an integer value. If this conversion fails, throw an exception. 3. Copy the key values from the original-request business object into the application-specific business object.

Because the original-request business object for ACCESS_RESPONSE is the application-specific business object, the Cross-Reference transformation automatically obtains this key value from the original-request business object in the map execution context (cwExecCtx).

The Cross-Reference transformation can perform the tasks in Table 78 as long as it has access to the original-request business object. However, in some cases, it might not have access to this business object. For example, if the Cross-Reference transformation is processing a child object that didn't exist in the primary request, the method tries to retrieve that child object's relationship instance ID. If the method can't find the relationship instance, it just populates the keys of this child object with the CxIgnore value.

Defining transformation rules for a simple identity relationship

For information on defining a Cross-Reference relationship, see "Cross-referencing identity relationships" on page 45.

Coding a child-level simple identity relationship

If child business objects have a unique key attribute, you can relate these child business objects in a simple identity relationship. Coding this simple identity relationship involves the following steps:

- "Creating the child relationship definition"
- "Customizing the parent map"
- "Customizing the submap" on page 202

Creating the child relationship definition: To create a relationship definition for a simple identity relationship between child business objects, take the following steps:

1. Create a participant definition whose participant type is the child business object.
2. Set the participant attribute to the key attribute of the child business object.

   Expand the child business object and select the key attribute.
3. Repeat steps 1 and 2 for each of the participants. As with all simple identity relationships, this relationship contains one participant for the generic business object and at least one participant for an application-specific business object. Each participant contains a single attribute: the key of the business object.

Customizing the parent map: In the map for the parent business object (the main map), add the mapping code to the attribute that contains the child business object. In the Activity Editor for this attribute, take the following steps to code a simple identity relationship:

1. If you created a submap for the child object, call this submap from the child attribute of the main map. Usually mapping transformations for a child object are done within a submap, especially if the child object has multiple cardinality.
2. Use the General/APIs/Identity Relationship/Maintain Child Verb function block to set the source child objects' verbs for you.

   The last parameter of the General/APIs/Identity Relationship/Maintain Child Verb function block is a boolean flag to indicate whether the child objects are participating in a composite relationship. Make sure you pass a value of false as the last argument to the General/APIs/Identity Relationship/Maintain Child Verb function block because this child object participates in a simple, not a composite identity relationship. If the child object has a submap, call the

General/APIs/Identity Relationship/Maintain Child Verb function block *before* the call to the submap. For more information, see "Setting the source child verb" on page 213.

Note: If the key attribute of the parent business object also participates in a simple identity relationship, define a Cross-Reference transformation in the main map, as described in "Cross-referencing identity relationships" on page 45.

Customizing the submap: In the submap, perform the following steps:

1. Define a More or Set Value transformation for the child business object.
2. Define a Cross-Reference transformation for the child business object and specify the relationship name and participant. For more information, see "Cross-referencing identity relationships" on page 45.

Using composite identity relationships

An identity relationship establishes an association between business objects or other data on a *one-to-one basis*. A composite identity relationship relates two business objects through a composite key attribute. The following sections describe the steps for working with composite identity relationships:

- "Creating composite identity relationship definitions"
- "Determining the relationship action" on page 203
- "Customizing map rules for a composite identity relationship" on page 204

Creating composite identity relationship definitions

Identity relationship definitions differ from lookup relationship definitions in that the participant types are business objects, *not* of the type `Data` (the first selection in the participant types list). As with a simple identity relationship, a composite identity relationship:

- The relationship consists of the generic business object and at least one application-specific business object.
- The participant type is a business object for *all* participants.

However, for a composite identity relationship, the participant attribute for every participant is a composite key. This composite key usually consists of a unique key from a parent business object and a nonunique key from a child business object.

To create a relationship definition for a composite identity relationship, take the following steps:

1. Create a participant definition whose participant type is the parent business object.
2. Set the first participant attribute to the key of the parent business object.

   Expand the parent business object and select the key attribute.
3. Set the second participant attribute to the key of the child attribute.

   Expand the parent business object, then expand the child attribute within the parent. Select the key attribute from this child object.
4. Repeat steps 1-3 for each of the participants. As with all composite identity relationships, this relationship contains one participant for the generic business object and at least one participant for a application-specific business object. Each participant consists of two attributes: the key of the parent business object and the key of the child business object (from the attribute within the parent business object).

Restriction: To manage composite relationships, the server creates internal tables. A table is created for each role in the relationship. A unique *index* is then created on these tables across all *key attributes* of the relationship. (In other words, the columns which correspond to the key attributes of the relationship are the participants of the index.) The column sizes of the internal tables have a direct relation to the attributes of the relationship and are determined by the value of the MaxLength attribute for the relationship.

Databases typically have restrictions on the size of the indexes that can be created. For instance, DB2 has an index limitation of 1024 bytes with the default page size. Thus, depending on the MaxLength attribute of a relationship and the number of attributes in a relationship, you could run into an index size restriction while creating composite relationships.

Important:

- You must ensure that appropriate MaxLength values are set in the repository file for all *key attributes* of a relationship, such that the total index would never exceed the index size limitations of the underlying DBMS.

  If the MaxLength attribute for type String is not specified, the default is nvarchar(255) in the SQLServer. Thus, if a relationship has *N* Keys, all of type String and the default MaxLength attribute of 255 bytes, the index size would be ((N*255)*2) + 16 bytes. You can see that you would exceed the SQLServer 7 limit of 900 bytes quite easily when N takes values of >=2 for the default MaxLength value of 255 bytes for type String.
- Remember, too, that even when some DBMS'es support large indexes, it comes at the cost of performance; hence, it is always a good idea to keep index sizes to the minimum.

For more information on how to create a relationship definition for a composite identity relationship, see "Defining identity relationships" on page 174.

Determining the relationship action

Table 79 shows the activity function blocks that the Mapping API provides to maintain a composite identity relationship from the child attribute of parent source business object. The actions that these methods take depends on the source object's verb and the calling context.

Table 79. Maintaining a composite identity relationship from the child attribute

Activity function blocks	Purpose
General/APIs/Identity Relationship/ Maintain Child Verb	Set source child verb correctly
General/APIs/Identity Relationship/ Maintain Composite Relationship	Perform appropriate action on the relationship tables

Actions of General/APIs/Identity Relationship/Maintain Composite Relationship

The Maintain Composite Relationship function block will generate Java code that calls the mapping API `maintainCompositeRelationship()`, which will manage relationship tables for a composite identity relationship. This method ensures that the relationship instances contain the associated application-specific key values for each relationship instance ID. This method automatically handles all of the basic adding and deleting of participants and relationship instances for a composite identity relationship.

The actions that maintainCompositeRelationship() takes are based on the value of the business object's verb and the calling context. The method iterates through the child objects of a specified participant, calling the maintainSimpleIdentityRelationship() on each one to correctly set the child key value. As with maintainSimpleIdentityRelationship(), the action that maintainCompositeRelationship() takes is based on the following information:

- The calling context: EVENT_DELIVERY, ACCESS_REQUEST, SERVICE_CALL_REQUEST, SERVICE_CALL_RESPONSE, SERVICE_CALL_FAILURE, and ACCESS_RESPONSE
- The verb of the source business object: Create, Update, Delete, or Retrieve

For information on the actions that maintainSimpleIdentityRelationship() takes, see "Accessing identity relationship tables" on page 191.

The maintainCompositeRelationship() method deals *only* with composite keys that extend to only two nested levels. In other words, the method cannot handle the case where the child object's composite key depends on values in it grandparent objects. For example, if A is the top-level business object, B is the child of A, and C is the child of B, the two methods will *not* support the participant definitions for the child object C that are as follows:

- The participant type is A and the attributes are:

```
key attribute of A: ID
key attribute of B: B[0].ID
key attribute of C: B[0].C[0].ID
```

- The participant type is A and the attributes are:

```
key attribute of A: ID
key attribute of C: B[0].C[0].ID
```

To access a grandchild object, these methods only support the participant definitions that are as follows:

- The participant type is B and the attributes are:

```
key attribute of B: ID
key attribute of C: C[0].ID
```

- The participant type is B and the attributes are:

```
key attribute of B: ID
first key attribute of C: C[0].ID1
second key attribute of C: C[0].ID2
```

Actions of General/APIs/Identity Relationship/Maintain Child Verb

The Maintain Child Verb function block will generate Java code that calls the mapping API maintainChildVerb(), which will maintain the verb of the child objects in the destination business object. It can handle child objects whose key attributes are part of a composite identity relationship. When you call maintainChildVerb() as part of a composite relationship, make sure that its last parameter has a value of true. This method ensures that the verb settings are appropriate given the verb in the parent source object and the calling context. For more information on the actions of maintainChildVerb(), see "Setting the source child verb" on page 213.

Customizing map rules for a composite identity relationship

Once you have created the relationship definition and participant definitions for the composite identity relationship, you can customize the map to maintain the composite identity relationship. A composite identity relationship manages a composite key. Therefore, managing this kind of relationship involves management

of both parts of the composite key. To code a composite identity relationship, you need to customize the mapping transformation rules for both the parent and child business objects, as Table 80 shows.

Table 80. Activity function blocks for a composite identity relationship

Map involved	Business object involved	Attribute	Activity function blocks
Main	Parent business object	Top-level business object	Use a Cross-Reference transformation rule
		Child attribute (child business object)	General/APIs/Identity Relationship/Maintain Composite Relationship General/APIs/Identity Relationship/Maintain Child Verb General/APIs/Identity Relationship/Update My Children (optional)
Submap	Child business object	Key attribute (nonunique key)	Define a Move or Set Value transformation for the verb.

If child business objects have a nonunique key attribute, you can relate these child business objects in a composite identity relationship. Customizing this composite identity relationship involves the following steps:

- "Customizing the main map"
- "Managing child instances" on page 207

Customizing the main map

In the map for the parent business object (the main map), add the mapping code to the following parent attributes:

- Map the verb of the top-level business object by defining a Move or Set Value transformation rule.
- Define a Cross-Reference transformation between the top-level business objects.
- Define a Custom transformation for the child attribute and use the General/APIs/Identity Relationship/Maintain Composite Relationship function block in Activity Editor.

Coding the child attribute: The child attribute of the parent object contains the child business object. This child object is usually a multiple cardinality business object. It contains a key attribute whose value identifies the child. However, this key value is not required to be unique. Therefore, it does not uniquely identify one child object among those for the same parent nor is it sufficient to identify the child object among child objects for all instances of the parent object.

To uniquely identify such a child object, the relationship uses a composite key. In the composite key, the parent key uniquely identifies the parent object. The combination of parent key and child key uniquely identifies the child object. In the map for the parent business object (the main map), add the mapping code to the attribute that contains the child business object. In the Activity Editor for this attribute, take the following steps to code a composite identity relationship:

1. Define a Submap transformation for the child business object attribute of the main map. Usually mapping transformations for a child object are done within a submap, especially if the child object has multiple cardinality.
2. In the main map, define a Custom transformation rule for the child verb and use the General/APIs/Identity Relationship/Maintain Child Verb function block to maintain the child business object's verb.

   The last input parameter of the General/APIs/Identity Relationship/Maintain Child Verb function block is a boolean flag to indicate whether the child objects

are participating in a composite relationship. Make sure you pass a value of `true` as the last argument to `maintainChildVerb()` because this child object participates in a composite, not a simple identity relationship. Make sure you call `maintainChildVerb()` *before* the code that calls the submap. For more information, see "Setting the source child verb" on page 213.

3. To maintain this composite key for the parent source object, customize the mapping rule to use the General/APIs/Identity Relationship/Maintain Composite Relationship function block.
4. To maintain the relationship tables in the case where a parent object has an Update verb caused by child objects being deleted, customize the mapping rule to use the General/APIs/Identity Relationship/Update My Children function block.

   Tip: Make sure the transformation rule that contains the Update My Children function block has an execution order after the transformation rule that contains the Maintain Composite Relationship function block.

Here is a sample of how the map can be customized for a Composite Identity Relationship.

1. In the main map, define a Custom transformation rule between the child business object's verbs. Use the General/APIs/Identity Relationship/Maintain Child Verb function block in the customized activity to maintain the verb for the child business objects.

   The goal of this custom activity is to use the maintainChildVerb() API to set the child business object verb based on the map execution context and the verb of the parent business object. Figure 111 shows this custom activity.

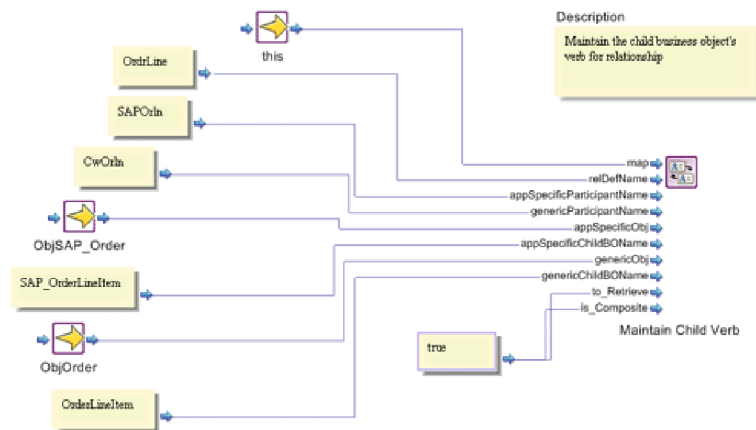


Figure 111. Using the Maintain Child Verb function block

2. If necessary, define a Submap transformation rule between the child business object to perform any mapping necessary in the child level.
3. Define a Custom transformation rule between the top-level business objects. Use the General/APIs/Identity Relationship/Maintain Composite Relationship function block in the customized activity to maintain the composite identity relationship for this map.

   The goal of this custom activity is to use the maintainComposite Relationship() API to maintain a compositie identity relationship within the map. Figure 112 on page 207

on page 207 shows this custom activity.

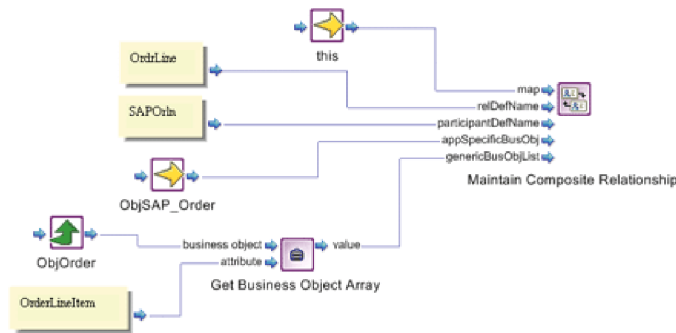


Figure 112. Using the Maintain Composite Relationship function block

4. Define a Custom transformation rule mapping from the source top-level business object to the destination child business object attribute. Use the General/APIs/Identity Relationship/Update My Children function block in the customized activity to maintain the child instances in the relationship.

   The goal of this custom activity is to use the updateMyChildren() API to add or delete child instances in the specified parent/child relationship of the identity relationship. Figure 113 shows this custom activity.

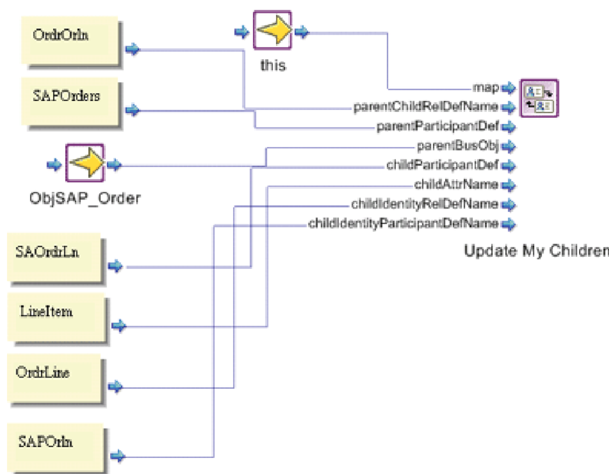


Figure 113. Using the Update My Children function block

Managing child instances

The Activity Editor provides the function blocks in Table 81 to manage child object instances that belong to a parent in an identity relationship.

Table 81. Function blocks for Managing Child Instances

Function block	Description
General/APIs/Identity Relationship/Add My Children	Adds child relationship instances to parent/child relationship tables
General/APIs/Identity Relationship/Delete My Children	Deletes child relationship instances to parent/child relationship tables

Table 81. Function blocks for Managing Child Instances (continued)

Function block	Description
General/APIs/Identity Relationship/Update My Children	Deletes or adds child relationship instances from parent/child relationship tables.

Note: The most common use of the function blocks in Table 81 is to maintain child business objects in custom relationships involving composite identity relationships.

The function blocks in Table 81 assume that the parent business object being passed is an after-image; that is, the image of the business object *after* the verb operation has taken place. For example, if a business object has an Update verb with the update caused by the addition of new child objects, these new child objects already exist in the business object. Similarly, if a business object has an Update verb with the update caused by the deletion of child objects, the business object already has these child objects deleted.

This section provides the following information about how to manage child instances:

- "Creating the parent/child relationship definition"
- "Handling updates to the parent business object" on page 209

Creating the parent/child relationship definition

A *parent/child relationship* is a 1-to-many relationship between parent (1) and child (many) business objects. A parent/child relationship involves the following participants:

- A participant containing the key attribute of that parent business object
- A participant containing the key of the child business object

The relationship tables for a parent/child relationship enable the function blocks in Table 81 to track the child business objects associated with a particular parent business object.

To create a relationship definition for a parent/child relationship, take the following steps in Relationship Designer Express:

1. Create a participant definition whose participant type is the parent business object.
2. Set the participant attribute to the key of the parent business object.

   Expand the parent business object and select the key attribute.
3. Create a participant definition whose participant type is the child business object.
4. Set the participant attribute to the key of the child attribute.

   Expand the child business object (not the child attribute with the parent object) and select the key attribute from this child object.

Note: The parent-child relationship needs to be maintained only if the child object does *not* have a unique key; that is, the child object *only* exists within the context of its parent.

For more information, see "Defining identity relationships" on page 174.

Handling updates to the parent business object

This section provides the following steps to ensure that child objects that participate in a composite identity relationship are correctly managed during an Update:

- "Comparing the before- and after-images"
- "Tips on using Update My Children"

Comparing the before- and after-images

The Update My Children function block updates the relationship tables for a parent/child relationship. A parent/child relationship is needed to help determine whether child objects have been added to or deleted from a parent business object.

For a given parent business object, this method makes sure that the following images of the business object match:

- The before-image information is contained in the relationship tables for the parent/child relationship.
- The after-image is contained in parent business object.

For the map to detect that a child business object has been deleted, it must determine how many instances of the child object of this type that the parent business object had before the Update (the before-image) and compare that to what the parent object presently has (the after-image). The map can use the Update My Children function block to make this comparison and find out what has been deleted or added.

When Update My Children compares the before- and after-images, it can determine whether to remove the associated relationship instances from the relationship tables for any child object that is *not* present in the parent business object. The method removed relationship instances from the following relationship tables:

- The relationship table for the child participant in the parent/child relationship
- The relationship table for the participant in the composite identity relationship that contains the parent and child objects

Note: Although Update My Children can also add instances to the relationship table for any child object that *is* present in the parent business object (but not in the child relationship table), it does not need to when called in the context of a composite identity relationship. All new child objects for the parent object have already been added to the relationship tables by the Maintain Composite Relationship function block For more information, see "Actions of General/APIs/Identity Relationship/Maintain Composite Relationship" on page 203.

Tips on using Update My Children

When you use the Update My Children function block to maintain relationship tables for a child object involved in a composite identity relationship, keep the following tips in mind:

- Make sure you use the Update My Children function block *after* the Maintain Composite Relationship function block and that you have set the appropriate verbs on the child business objects.
- The Update My Children function block is only needed to track child objects involved in composite relationships.

You do *not* need to use the Update My Children function block to track child objects involved in a simple identity relationship. For more information, see "Coding a child-level simple identity relationship" on page 201.

- The Update My Children function block (as with the Maintain Composite Relationship function block) deals only with composite keys that extend to only two nested levels: the parent and its immediate children.

  In other words, the method cannot handle the case where the grandchild object's composite key depends on values in it grandparent objects. For example, if A is the top-level business object, B is the child of A, and C is the child of B, the two methods will not support the participant definitions for the child object C that are as follows:

  – The participant type is A and the attributes are:

  ```
  key attribute of A: ID
  key attribute of B: B[0].ID
  key attribute of C: B[0].C[0].ID
  ```

  – The participant type is A and the attributes are:

  ```
  key attribute of A: ID
  key attribute of C: B[0].C[0].ID
  ```

  To access a grandchild object, these methods only support the participant definitions that are as follows:

  – The participant type is B and the attributes are:

  ```
  key attribute of B: ID
  key attribute of C: C[0].ID
  ```

  – The participant type is B and the attributes are:

  ```
  key attribute of B: ID
  first key attribute of C: C[0].ID1
  second key attribute of C: C[0].ID2
  ```

- The Update My Children function block manages the parent/child relationship tables for the `EVENT_DELIVERY` and `SERVICE_CALL_RESPONSE` calling contexts *only*.

  Execution of the Update My Children function block with a calling context of `SERVICE_CALL_REQUEST` or `ACCESS_RESPONSE` does *not* produce any changes to these relationship tables.

- The Update My Children function block can also be used when the child business object has a unique ID; that is, the child object participates in a simple identity relationship. In this case, you must still define the parent/child relationship (see "Creating the parent/child relationship definition" on page 208).

Setting the verb

This section contains the following information on how to set the verb of a business object participating in a map:

- "Conditionally setting the destination verb"
- "Setting the source child verb" on page 213

Note: For general information about how to set the verb of the destination business object, see "Setting the destination business object verb" on page 35.

Conditionally setting the destination verb

Usually, you just set the destination verb to the value of the source verb by defining a Move transformation. (For more information on this action, see "Setting

the destination business object verb" on page 35.) However, sometimes the source application sets the business object verb in an unusual manner; for example, the verb is set to Update even though the event is new. As another example, the verb is always set to Retrieve. In the situations like these, the map must resets the destination verb to the one that corresponds to the actual event.

If the source business object's key participates in a relationship, the map can perform a *static lookup* in the relationship table to determine if the source business object exists. The map can then set the destination verb to either Update or Create based on whether the corresponding entry is found in the table. You perform this static lookup in much the same way as accessing a lookup relationship. Table 82 shows the function block to use for each kind of static lookup.

Table 82. Checking for Existence of the source business object

Type of source business object	Map type	Function block
Application-specific	Inbound	General/APIs/Relationship/ Retrieve Instances
Generic	Outbound	General/APIs/Relationship/ Retrieve Participants

Example of customizing the inbound map

Here is an example of how an inbound map can conditionally set the destination verb based on the result of a lookup:

1. In the map, define a Custom transformation between the source business object and the destination verb.
2. In the Activity of this Custom Transformation, perform the following steps. The goal of this activity is to identify the number of instances in the participant of the relationship. If there are no participant instances in the relationship, the destination business object verb should be Create; otherwise, the verb should be Update.
   a. Define the activity, as shown in Figure 114, to identify the number of instances in the relationship participant.

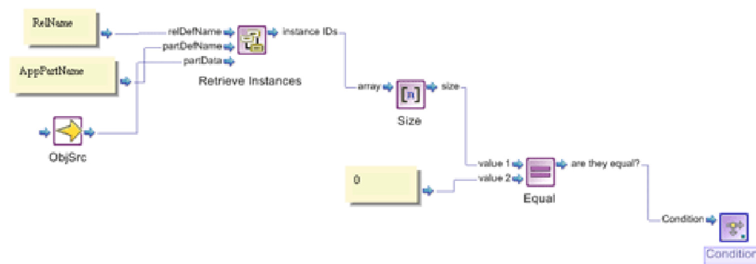


Figure 114. Identifying the number of instances in the relationship participant

   b. Double-click the Condition function block in the canvas to open it. Select True Action to define the action to take when the condition is true. Define

the True Action as shown in Figure 115.

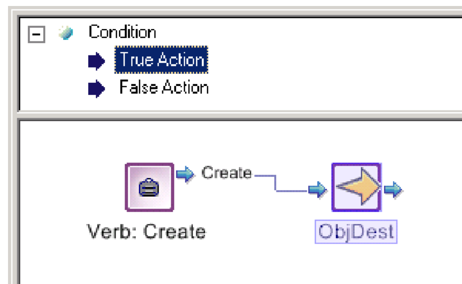


Figure 115. Defining the True Action

c. Select the False Action to define the action to take when the number of participant instances is not zero. Define the False Action as shown in Figure 116.

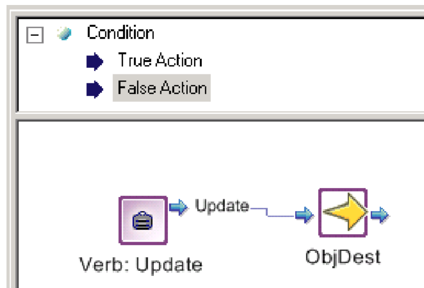


Figure 116. Defining the False Action

Example of customizing the outbound map

You can use similar steps in the outbound map to perform a static lookup based on the primary key of the generic object. To do that, you need to replace the function block General/APIs Relationship/Retrieve Instances with the function block General/APIs Relationship/Retrieve Participants. Here are the steps:

1. In the map, define a Custom transformation between the key attribute of the source business object and the destination verb.
2. In the activity of this Custom transformation, perform the following steps. The goal of this activity is to identify the number of participants of the relationship. If there are no participant instances in the relationship the destination business object verb should be Create; otherwise, the verb should be Update.

a. Define the activity, as shown in Figure 117, to identify the number of participants in the relationship.

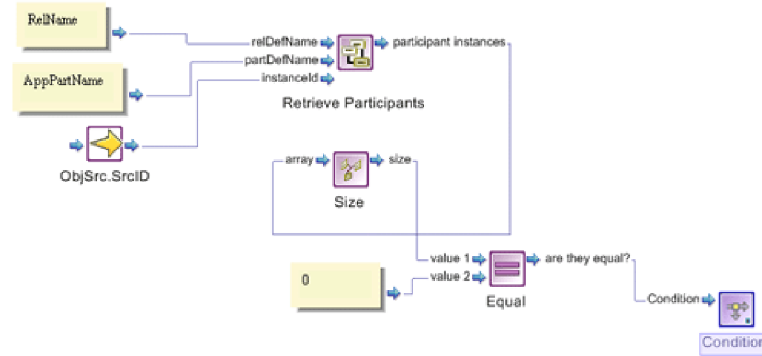


Figure 117. Identifying the number of participants in the relationship

b. Follow steps 2b and 2c, described in "Example of customizing the inbound map" on page 211.

Setting the source child verb

When a parent source business object has child business objects, the value of the source child verb is usually the same as that of the parent verb. Therefore, you set the source child object's verb by defining a Move transformation from the parent verb to the child verb. However, if the parent object's verb is Update, the update could be a result of any of the modifications shown in Table 83.

Table 83. Updating a parent business object

Update task	Verb in child object
Modifying some non-child attribute i the parent object	Update
Modifying some attribute in a child object	Update
Adding more child objects	Create
Deleting existing child objects	Delete

All of the modifications Table 83 are represented by a verb of Update in the parent object. However, not all of these modifications represent an Update to the child object. The value of the source child verb depends on what action was taken on the parent verb. When the child object's key participates in an identity relationship (composite or simple), the source child verb value depends not just on the parent verb but also on the calling context. In such cases, use the Maintain Child Verb function block to handle the setting of the verb of the source child object.

This section provides the following information about using the Maintain Child Verb function block to maintain a source child object verb:

- "Determining the child verb setting"
- "Tips for using the Maintain Child Verb function block" on page 215

Determining the child verb setting

The Maintain Child Verb function block must ensure that the verb settings of the child objects in the source business object are appropriate given the verb in the parent source object and the calling context. The actions that this method takes are based on the verb in the parent source object and the calling context.

EVENT_DELIVERY and ACCESS_REQUEST calling Contexts: When the calling context is `EVENT_DELIVERY` or `ACCESS_REQUEST`, the map is being called is an inbound map; that is, it transforms an application-specific business object to a generic business object. The inbound map receives an application-specific business object as input and returns a generic business object as output. For `EVENT_DELIEVERY` (or `ACCESS_REQUEST`), there are no special cases to handle when setting the child verbs. Therefore, the `maintainChildVerb()` method just copies the parent verb to the child verb for all verb values, as Table 84 shows.

Table 84. Actions for the EVENT_DELIVERY and ACCESS_REQUEST calling contexts

Verb of generic business object	Action performed by the Maintain Child Verb function block
Create Delete Update Retrieve	Set the verbs of all child objects in the source object to the verb in the parent source object. This action overwrites any existing verb in the child object.

SERVICE_CALL_REQUEST calling context: When the calling context is `SERVICE_CALL_REQUEST`, the map is being called is an outbound map; that is, it transforms a generic business object to an application-specific business object. The outbound map receives a generic business object as input and returns an application-specific business object as output. For `SERVICE_CALL_REQUEST`, the Java code generated by the Maintain Child Verb function block handles the special case for an Update verb: If the change to the parent object is the creation of new child objects, the Maintain Child Verb function block changes the verb to Create for any child objects that do not currently exist in the relationship tables, as Table 85 shows.

Table 85. Actions for SERVICE_CALL_REQUEST calling context

Verb of generic business object	Action performed by the Maintain Child Verb function block
Create Delete Retrieve	Set the verbs of all child objects in the source object to the verb in the parent source object. This action overwrites any existing verb in the child object.
Update	1. Retrieve the relationship instance from the child relationship table for the given generic business object's key value. 2. Set the verb of the child object based on the success of the table lookup: • If a relationship instance for this child object exists, set the verb of the child object to Update. • If a relationship instance for this child object does *not* exist, set the verb of the child object to Create.

SERVICE_CALL_RESPONSE calling context: When the calling context is `SERVICE_CALL_RESPONSE`, the map is being called is an inbound map; that is it transforms an application-specific business object to a generic business object. The inbound map receives an application-specific business object as input and returns a generic business object as output.

The behavior of the Maintain Child Verb function block is determined by the second-to-last parameter of the method. This parameter is the boolean *`to_Retrieve`* flag, whose value indicates whether the application resets or preserves child objects' verbs when processing a collaboration request, as Table 86 shows.

Table 86. Connector behavior

Value of to_Retrieve flag	Connector behavior
`true`	Connector sets child object verbs to different value from what they had coming into the application. For example, if a business object comes to the connector with a parent verb of Update and a child verb of Create, the connector might reset all child object verbs to their parent value after the application completes the operation. In this case, the child verb would be changed to Update.
`false`	Connector preserves child object verbs. For example, if a business object comes to the connector with a parent verb of Update and a child verb of Create, the connector preserves all child object verbs. In this case, the child verb would still be Create.

Note: The Java code generated by the Maintain Child Verb function block uses the value of the *`to_Retrieve`* parameter *only* when it processes the `SERVICE_CALL_RESPONSE` calling context.

If the *`to_Retrieve`* argument is `true`, the Maintain Child Verb function block performs the tasks in Table 87..

Table 87. Actions for the SERVICE_CALL_RESPONSE calling context

Verb of generic business object	Action performed by the Maintain Child Verb function block
Create Delete Retrieve Update	Set the verbs of all child objects in the source object to the verb in the parent source object. This action overwrites any existing verb in the child object. 1. Lookup each child object in the child relationship table. 2. Set the verb of the child object based on the success of the table lookup: • If a relationship instance for this child object exists, set the verb of the child object to Update. • If a relationship instance for this child object does *not* exist, set the verb of the child object to Create.

Note: If you are unsure of the behavior of your application, set the *`to_Retrieve`* argument to `true`. With a `true` flag value, performance might be affected because the Java code generated by the Maintain Child Verb function block might perform an unnecessary lookup. However, it is usually safer to have an unnecessary lookup than to have an incorrect verb setting in the child object.

Tips for using the Maintain Child Verb function block

The Maintain Child Verb function block maintains the verb of the child objects in the source business object. It can handle child objects that are part of a simple or a composite identity relationship. This function block must ensure that the verb settings are appropriate given the verb in the parent source object and the calling context.

Keep the following tips in mind when using the Maintain Child Verb function block:

- The second to last parameter in this method is the *`to_Retrieve`* boolean flag, which indicates whether the application resets or preserves child objects' verbs.

For more information on how to set the *to_Retrieve* flag, see "SERVICE_CALL_RESPONSE calling context" on page 214.

- The last parameter in this method is the *is_Composite* boolean flag, which indicates whether the child object is part of a simple or composite identity relationship.

  The key attribute of a child business object can participate in either of the following kinds of identity relationship:

  – As a unique key in a simple identity relationship

    Set the value of the *is_Composite* flag to `false`.

  – As a nonunique key of a composite key in a composite identity relationship; in this case, the other part of the composite key is the unique key in the parent business object.

    Set the value of the *is_Composite* flag to `true`.

- Make sure you use the Maintain Child Verb function block in the child attribute of the source parent map, *before* calling the submap.

  For multiple-cardinality child objects, use the Maintain Child Verb function block right *before* the start of the `for` loop. The method iterates through the child objects to set the child verbs correctly.

Performing foreign key lookups

A *foreign key* is an attribute within one business object that contains the key value of another business object. This key value is considered "foreign" to the source business object because it identifies some other business object. To transform a foreign key in a source business object, you must access the relationship table associated with the business object that the foreign key references (the foreign relationship table). From this foreign relationship table, you can obtain the associated key value for the foreign key of the destination business object.

The Mapping API provides the methods in Table 88 to perform foreign key lookups.

Table 88. Function blocks for foreign key lookups

Function block	Description
General/APIs/Identity Relationship/Foreign Key Lookup	Performs a foreign key lookup, failing to find a relationship instance if the foreign key does not exist in the foreign relationship table.
General/APIs/Identity Relationship/Foreign Key Cross-Reference	Performs a foreign key lookup, adding a new relationship instance in the foreign relationship table if the foreign key does not exist.

Using the Foreign Key Lookup function block

The Java code generated by the Foreign Key Lookup function block performs a lookup in a foreign relationship table for the foreign key of the source business object. This function block takes the following actions:

1. Verify that the application-specific participant contains a single key, *not* a composite key.

   Determine the participant type of the application-specific participant, which is the application-specific business object. In this business object, verify that only one key attribute exists. If more than one key attribute exists, the Foreign Key Lookup function block does not know which application-specific key attribute

to populate with the application-specific equivalent of the generic business object's foreign key. Therefore, it throws the `RelationshipRuntimeException` exception.

2. Locate the relationship instance in the foreign relationship table that matches the value of the foreign key in the generic business object.
3. Obtain the application-specific key value from the retrieved relationship instance.
4. Copy the application-specific key value into the foreign key of the application-specific business object.

The Java code generated by the Foreign Key Lookup function block takes these actions on the foreign relationship table regardless of the verb in the source business object.

Using the Foreign Key Cross-Reference function block

As with the Foreign Key Lookup function block, the Foreign Key Cross-Reference function block performs a lookup in a foreign relationship table based on the foreign key of the source business object. However, the Foreign Key Cross-Reference function block provides the additional functionality that it can add an entry to the foreign relationship table if the lookup fails. The following sections discuss the behavior of the Foreign Key Cross-Reference function block with each of the calling contexts.

EVENT_DELIVERY, ACCESS_REQUEST, and SERVICE_CALL_RESPONSE calling contexts

When the calling context is `EVENT_DELIVERY`, `ACCESS_REQUEST`, or `SERVICE_CALL_RESPONSE`, the map is being called is an inbound map; that is, it transforms an application-specific business object to a generic business object. The inbound map receives an application-specific business object as input and returns a generic business object as output. Therefore, the task for the Foreign Key Cross-Reference function block is to obtain from the foreign relationship table the generic key for a given application-specific key value.

For the `EVENT_DELIVERY`, `ACCESS_REQUEST`, and `SERVICE_CALL_RESPONSE` calling contexts, the Foreign Key Cross-Reference function block takes the following actions:

1. Verify that the generic participant contains a single key, *not* a composite key.

   Determine the participant type of the generic participant, which is the generic business object. In this business object, verify that only one key attribute exists. If more than one key attribute exists, the Foreign Key Cross-Reference function block does not know which generic key attribute to populate with the generic equivalent of the application-specific business object's foreign key. Therefore, it throws the `RelationshipRuntimeException` exception.
2. Locate the relationship instance in the foreign relationship table that matches the value of the foreign key in the application-specific business object. Table 89 shows the actions that the Foreign Key Cross-Reference function block takes on the foreign relationship table based on the verb of the application-specific business object.
3. Obtain the instance ID from the retrieved relationship instance.

4. Copy the instance ID into the foreign key of the generic business object.

Table 89. Actions for EVENT_DELIVERY, ACCESS_REQUEST, and SERVICE_CALL_RESPONSE

Verb of application-specific business object	Action performed by the Foreign Key Cross-Reference function block
Create	For the EVENT_DELIVERY and ACCESS_REQUEST calling contexts, insert a new relationship entry into the foreign relationship table for the application-specific business object's key value. For the SERVICE_CALL_RESPONSE calling context, insert into the relationship table the new relationship entry containing the application-specific business object's key value and its associated relationship instance ID. The method obtains the relationship instance ID from the original-request business object in the map execution context (cwExecCtx). For more information on the behavior of the SERVICE_CALL_RESPONSE, see "SERVICE_CALL_RESPONSE calling context" on page 197. If an entry for this key value already exists, retrieve the existing one; do *not* add another one to the table.
Update	Retrieve the relationship entry from the foreign relationship table for the given application-specific business object's foreign key value. If an entry for this foreign key value does *not* exist, insert a new relationship instance into the foreign relationship table for the application-specific business object's foreign key value.
Retrieve	Retrieve the relationship entry from the foreign relationship table for the given application-specific business object's foreign key value

Figure 118 shows how the Foreign Key Cross-Reference function block accesses the foreign relationship table (for App Obj C) when a calling context is EVENT_DELIVERY, ACCESS_REQUEST, or SERVICE_CALL_RESPONSE and the verb for the application-specific business object (App Obj A) is either Create or Update.

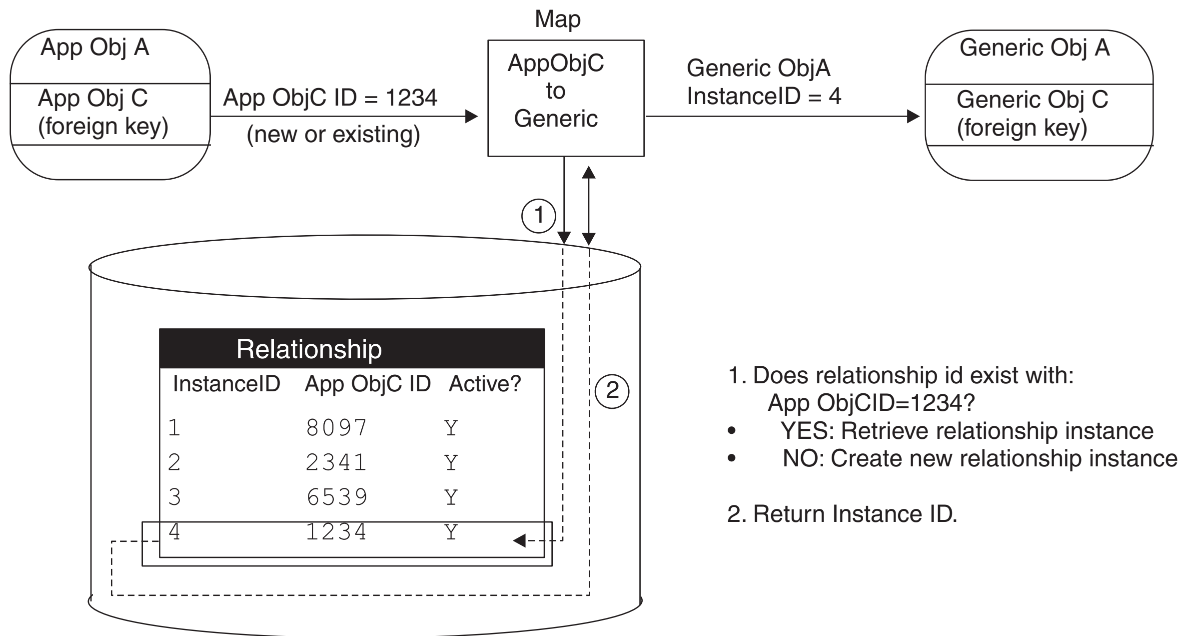


Figure 118. Foreign key lookup for a create or update verb

Note: The Foreign Key Cross-Reference function block only adds relationship instances to the foreign relationship table for inbound maps.

SERVICE_CALL_REQUEST calling context and Foreign Keys

When the calling context is `SERVICE_CALL_REQUEST`, the map is being called is an outbound map; that is, it transforms a generic business object to an application-specific business object. The outbound map receives a generic business object as input and returns an application-specific business object as output. For the `SERVICE_CALL_REQUEST` calling context, the Foreign Key Cross-Reference function block takes the following actions:

1. Verify that the application-specific participant contains a single key, *not* a composite key.

   Determine the participant type of the application-specific participant, which is the application-specific business object. In this business object, verify that only one key attribute exists. If more than one key attribute exists, the Foreign Key Cross-Reference function block does not know which application-specific key attribute to populate with the application-specific equivalent of the generic business object's foreign key. Therefore, it throws the `RelationshipRuntimeException` exception.
2. Perform the task outlined in Table 90, based on the verb of the application-specific business object.

   The Foreign Key Cross-Reference function block obtains from the foreign relationship table an application-specific business object's key value for a given a relationship instance ID *only* if the verb is Update, Delete, or Retrieve. The Foreign Key Cross-Reference function block does *not* obtain the application-specific key value for a Create verb.

Table 90 shows the action that the Foreign Key Cross-Reference function block takes on the foreign relationship table based on the verb of the generic business object.

Table 90. Actions for the SERVICE_CALL_REQUEST calling context and a Foreign Key

Verb of generic business object	Action performed by the Foreign Key Cross-Reference function block
Create	Take no action. The method writes a new relationship instance to the foreign relationship table when the calling context is SERVICE_CALL_RESPONSE. For more information, see "EVENT_DELIVERY, ACCESS_REQUEST, and SERVICE_CALL_RESPONSE calling contexts" on page 217.
Update Delete Retrieve	1. Obtain the generic business object's key value (the relationship instance ID) from the original-request business object in the map execution context. 2. Retrieve the relationship instance from the foreign relationship table for the given generic business object's key value. If a relationship instance for this key value does *not* exist, throw a RelationshipRuntimeException exception. If no participants are found when the verb is Retrieve, throw a CxMissingIDException exception. 3. Obtain the application-specific key value from the retrieved relationship instance. 4. Copy the application-specific key value into the application-specific business object.

As Table 90 shows, when the verb is Create, the Foreign Key Cross-Reference function block does *not* write a new relationship instance to the relationship table. It does not perform this write operation because it does not yet have the application-specific foreign key value that corresponds to the instance ID. When the connector processes the application-specific business object, it notifies the application of the need to insert a new row (or rows). If this insert is successful, the application notifies the connector, which creates the appropriate application-specific business object with a Create verb and the application's key value.

Note: For the SERVICE_CALL_REQUEST calling context, the Foreign Key Cross-Reference function block manages the foreign relationship table in the same way that the Maintain Simple Identity Relationship function block manages a relationship table.

ACCESS_RESPONSE calling context and foreign keys

When the calling context is ACCESS_RESPONSE, the map is being called is an outbound map; that is, it transforms a generic business object to an application-specific business object. The outbound map receives a generic business object as input and returns an application-specific business object as output. Therefore, the task for the Foreign Key Cross-Reference function block is to obtain from the foreign relationship table the application-specific key for a given generic key value.

For the ACCESS_RESPONSE calling context, the Foreign Key Cross-Reference function block takes the following actions:

1. Verify that the application-specific participant contains a single key, *not* a composite key.

   Determine the participant type of the application-specific participant, which is the application-specific business object. In this business object, verify that only one key attribute exists. If more than one key attribute exists, the Foreign Key Cross-Reference function block does not know which application-specific key attribute to populate with the application-specific equivalent of the generic business object's foreign key. Therefore, it throws the RelationshipRuntimeException exception.
2. Locate the relationship instance in the foreign relationship table that matches the value of the foreign key in the generic business object.

3. Obtain the application-specific key value from the retrieved relationship instance.
4. Copy the application-specific key value into the foreign key of the application-specific business object.

The Foreign Key Cross-Reference function block takes these actions on the foreign relationship table regardless of the verb in the generic business object.

Tips for using the Foreign Key Cross-Reference and Foreign Key Lookup function blocks

Keep the following tips in mind when using the Foreign Key Cross-Reference and Foreign Key Lookup function blocks:

- Put the call to the Foreign Key Lookup or Foreign Key Cross-Reference function blocks in the transformation step for the foreign key attribute of the destination business object.
- The Foreign Key Lookup and Foreign Key Cross-Reference function blocks do *not* support composite keys as the foreign key.
- After using the Foreign Key Lookup function block, check that the destination foreign key attribute does *not* contain a null value. A null foreign key value indicates that the Foreign Key Lookup function block was not able to locate the corresponding foreign key value for the foreign key in the source business object. To indicate this condition, log message number 5007 or 5008 (depending on whether or not the map is forced to fail) and, optionally, throw the `MapFailureException` exception to stop the map.

  You do *not* need this check after using Foreign Key Cross-Reference function block because this function block automatically adds an entry to the foreign relationship table if the application-specific key value does not exist.
- If any of the child object attributes require the use of the Foreign Key Cross-Reference function block or the Foreign Key Lookup function block (but not the Maintain Simple Identity Relationship function block or the Maintain Composite Relationship function block), you can set the verb of the source child object by defining a Move transformation from the source parent object's verb to the child business object's verb. Make the call *inside* the `for` loop, just before the `runMap()` method is called.

Loading and unloading relationships

With the `repos_copy` utility, you can load and unload specified relationship definitions in the repository.

Note: You can also use `repos_copy` to load and unload map definitions in the repository. For more information, see "Importing and exporting maps from InterChange Server Express" on page 67..

Unloading a relationship definition

With the `repos_copy` utility, you can unload specified relationship definitions in the repository with the `-e` option. A *relationship repository file* is the file that the `repos_copy` utility creates when it extracts a relationship definition from the repository into a text (`.jar`) file.

For example, the following `repos_copy` command unloads the `StateLk` relationship definition from the repository of an InterChange Server Express named `dexter` into a relationship repository file:

```
repos_copy -eRelationship:StateLk -oRL_StateLookup.jar
-sdexter -uadmin -pnull
```

Attention: A relationship is *not* a first-class entity. Therefore, its name space is separate from the first-class entities. While no first-class entities can have the same name, a relationship can have the same name as a first-class entity (such as a business object or collaboration). However, if a relationship definition has a name that matches *any* existing first-class entity, you cannot use the `-e` option of `repos_copy` to unload or load that relationship definition. You can load and unload the entire repository, which includes relationship definitions.

You can copy several the relationship definitions into one relationship repository file. For example, to copy both the `StateLk` and `CustLkUp` relationship definitions, use the following `repos_copy` command:

```
repos_copy –eRelationship:StateLk+Relationship:CustLkUp
–oRL_Lookup_Relationships.jar –sdexter -uadmin -pnull
```

Loading a relationship definition

You can also use `repos_copy` to load a relationship definition into the repository from a relationship repository file. The following `repos_copy` command loads the `StateLk` relationship definition into the repository of an InterChange Server Express named `testing`:

```
repos_copy -iRL_StateLookup.jar -stesting -uadmin -pnull
```

The `repos_copy` utility performs the following validations when it loads a relationship definition:

- It validates the Database URL of the relationship definition it loads.
- It validates that any dependent objects for the relationship definition already exist in the repository.

If `repos_copy` cannot perform both of these validations, it cannot load the relationship definition. However, `repos_copy` provides special command-line options to suppress or restrict these validations, as the following sections explain.

Validating the database URL

The `repos_copy` utility provides the `-r` option to assist in loading relationship definitions into a repository. The `-r` option tells `repos_copy` to add relationship definitions to the repository without creating their run-time schemas. When `repos_copy` backs up an entire repository (with the `-o` option), some of the information in the resulting repository text file describes relationship definitions. If you then use `repos_copy` (without the `-r` option) to load a different repository with the contents of this repository text file, `repos_copy` might generate errors of the following format when it attempts to load the relationship definitions:

```
Server error: An error occurred during the validation of the runtime database
connection information for relationship definition Customer. The database URL
used is: jdbc:weblogic:mssqlserver4:Cwrelns312@CWDEV:1433. The database
login name used is: crossworlds. The database type used is: W55s/wPE/l4=1.
Reason: SqlServer.
```

The cause of this error is `repos_copy`'s attempt to validate the URL for the relationship database. Part of a relationship's definition is the Database URL of the relationship database.

If `repos_copy` cannot find the relationship database, it generates an error and rolls back the repository load. If you are just backing up and restoring on the same

InterChange Server Express (with the same relationship databases), you do not need to include the -r option. Validation of the relationship database URL succeeds because the database URLs can be located. Therefore, the repository load (including the relationship definitions) is successful.

However, in the import process of a migration when you are moving repository data from one machine to another, the -r option can be helpful. If you execute the repos_copy command in an environment that cannot locate any existing relationship databases in the repository data, repos_copy generates the validation error. To suppress this validation, include the -r option of repos_copy when you load the repository. By suppressing this validation, repos_copy can successfully add the relationship definitions to the repository. It uses the current repository database as the location for the relationship database. You can then use Relationship Designer Express to change the Database URL to point to the appropriate location of each relationship database.

The following repos_copy command loads the StateLk relationship definition into the repository, suppressing the validation of its Database URL:

```
repos_copy -rStateLk -iRL_StateLookup.txt -stesting -uadmin
-pnull
```

Validating dependent objects

By default, repos_copy validates whether all dependent objects exist when it loads a relationship definition. For example, it checks that all business objects involved in the relationship exist in the repository. If all dependent objects do *not* exist, repos_copy generates an error and rolls back the repository load. In the repos_copy command window, the following message is displayed:

```
Some of the participants for relationships were missing.
For more info, refer to InterChange Server Express log file.
```

Part 3. Mapping API Reference

© Copyright IBM Corp. 2003

Chapter 9. BaseDLM class

The methods documented in this chapter operate on map instances. They are defined on the IBM WebSphere InterChange Server Express-defined class BaseDLM. The BaseDLM class is the base class for all map instances. All created maps are subclasses of BaseDLM; they all inherit these methods. The BaseDLM class provides utility methods for error handling and debugging in maps, and establishing a connection to a database. All methods in this class can be called without referring to the class name.

Table 91 summarizes the methods of the BaseDLM class.

Table 91. BaseDLM method summary

Method	Description	Page
getDBConnection()	Establishes a connection to a database and returns a CwDBConnection object.	227
getName()	Retrieves the name of the current map.	229
getRelConnection()	Establishes a connection to a relationship database and returns a DtpConnection object.	230
implicitDBTransactionBracketing()	Retrieves the transaction programming model that the map instance uses for any connection it obtains.	231
isTraceEnabled()	Compares the specified trace level with the current trace level of the map.	231
logError(), logInfo(), logWarning()	Sends an error, information, or warning message to the InterChange Server log file.	232
raiseException()	Raises an exception.	233
releaseRelConnection()	Releases a connection to a relationship database.	235
trace()	Generates a trace message.	236

getDBConnection()

Establishes a connection to a database and returns a CwDBConnection object.

Syntax

```
CwDBConnection getDBConnection(String connectionPoolName)
CwDBConnection getDBConnection(String connectionPoolName,
   boolean implicitTransaction)
```

Parameters

connectionPoolName
: The name of a valid connection pool. The method connects to the database whose connection is in this specified connection pool.

implicitTransaction
: A boolean value to indicate the transaction programming model to use for the database associated with the connection. Valid values are:

true	Database uses implicit transaction bracketing
false	Database uses explicit transaction bracketing

© Copyright IBM Corp. 2003

Return values

Returns a CwDBConnection object.

Exceptions

CwDBConnectionFactoryException – If an error occurs while trying to establish the database connection.

Notes

The getDBConnection() method obtains a connection from the connection pool that *connectionPoolName* specifies. This connection provides a way to perform queries and updates to the database associated with that connection. All connections in a particular connection pool are associated with the same database. The method returns a CwDBConnection object through which you can execute queries and manage transactions on the database. See the methods in the CwDBConnection class for more information.

By default, all connections use implicit transaction bracketing as their transaction programming model. To specify a transaction programming model *for a particular connection*, provide a boolean value to indicate the desired transaction programming model as the optional *implicitTransaction* argument to the getDBConnection() method. The following getDBConnection() call specifies explicit transaction bracketing for the connection obtained from the ConnPool connection pool:

```
conn = getDBConnection("ConnPool",false);
```

The connection is released when the map instance finishes execution. You can explicitly close this connection with the release() method. You can determine whether a connection has been released with the isActive() method.

Examples

The following example establishes a connection to the database associated with connections in the CustConnPool connection pool. It then uses an implicit transaction to insert and update rows in a table of the database.

```
CwDBConnection connection = getDBConnection("CustConnPool");

// Insert a row
connection.executeSQL("insert...");

// Update rows...
connection.executeSQL("update...");
```

Because the preceding call to getDBConnection() does *not* include the optional second argument, this connection uses implicit transaction bracketing as its transaction programming model (unless the transaction programming model is overridden in the Map Properties dialog). Therefore, it does not specify explicit transaction boundaries with beginTransaction(), commit(), and rollback(). In fact, an attempt to call one of these transaction methods with implicit transaction bracketing generates a CwDBTransactionException exception.

Note: You can check the current transaction programming model with the implicitDBTransactionBracketing() method.

The following example also establishes a connection to the database associated with connections in the CustConnPool connection pool. However, it specifies the

use of explicit transaction bracketing for the connection. Therefore, it uses an explicit transaction to contain the inserts and updates on rows in the database tables.

```
CwDBConnection connection = getDBConnection("CustConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Update rows...
connection.executeSQL("update...");

// Commit the transaction
connection.commit();

// Release the connection
connection.release();
```

The preceding call to `getDBConnection()` includes the optional *`implicitTransaction`* argument to set the transaction programming model to explicit transaction bracketing. Therefore, this examples uses the explicit transaction calls to indicate the boundaries of the transaction. If these transaction methods are omitted, InterChange Server Express handles the transaction as it would for an implicit transaction.

See also

`Chapter 12, "CwDBConnection class"`, `implicitDBTransactionBracketing()`, `isActive()`, `release()`

---

getName()

Retrieves the name of the current map.

Syntax

```
String getName()
```

Parameters

None.

Return values

None.

Exceptions

None.

Examples

The following example obtains the name of the current map and logs an informational message:

```
String mapName = getName();
logInfo(mapName + " is starting");
```

getRelConnection()

Establishes a connection to a relationship database and returns a `DtpConnection` object.

Syntax

```
DtpConnection getRelConnection(String relDefName)
```

Parameters

relDefName — A relationship definition name. The method connects to the database containing the relationship tables for this relationship definition.

Return values

Returns a `DtpConnection` object.

Exceptions

`DtpConnectionException` – If an error occurs while trying to establish the database connection.

Notes

This method establishes a connection to the database that contains the relationship tables used by the *relDefName* relationship, and provides a way to perform queries and updates to the relationship database. The method returns a `DtpConnection` object through which you can execute queries and manage transactions. See the methods in the `DtpConnection` class for more information.

The connection is released when the map is finished executing. You can explicitly close this connection with the `releaseRelConnection()` method.

Examples

The following example establishes a connection to the database containing the relationship tables for the `SapCust` relationship. It then uses a transaction to execute a query for inserting rows into a table in the `SapCust` relationship.

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// update rows...
connection.executeSQL("update...");

// commit the transaction
connection.commit();
```

See also

`getDBConnection()`, Chapter 14, "DtpConnection class", `releaseRelConnection()`

implicitDBTransactionBracketing()

Retrieves the transaction programming model that the map instance uses for any connection it obtains.

Syntax

```
boolean implicitDBTransactionBracketing()
```

Parameters

None.

Return values

A `boolean` value to indicate the transaction programming model to be used in all database connections.

Notes

The `implicitDBTransactionBracketing()` method returns a boolean value indicates which transaction programming model the map instance assumes is used by *all* connections that it obtains, as follows:

- A value of `true` indicates that all connections use *implicit* transaction bracketing.
- A value of `false` indicates that all connections use *explicit* transaction bracketing.

This method is useful before obtaining a connection to see whether the current transaction programming model is appropriate for that connection.

Note: You can override the transaction programming model for a particular connection with the `getDBConnection()` method.

Examples

The following example ensures that map instance uses explicit transaction bracketing for the database associated with the `conn` connection:

```
if (implicitDBTransactionBracketing())
   CwDBConnection conn = getDBConnection("ConnPool", false);
```

See also

`getDBConnection()`

isTraceEnabled()

Compares the specified trace level with the current trace level of the map.

Syntax

```
Boolean isTraceEnabled(int traceLevel)
```

Parameters

traceLevel The trace level to compare with the current trace level.

Return values

Returns `true` if the current system trace level is set to the specified trace level; returns `false` if the two trace levels are not the same.

Notes

The `isTraceEnabled()` method is useful in determining whether or not to log a trace message. Because tracing can decrease performance, this method is useful in the development phase of a project.

Examples

```
if ( isTraceEnabled(3) )
   {
   trace("Print this level-3 trace message");
   }
```

logError(), logInfo(), logWarning()

Sends an error, information, or warning message to the InterChange Server log file.

Syntax

```
void logError(String message)
void logError(int messageNum)
void logError(int messageNum, String param [,...])
void logError(int messageNum, Object[] paramArray)

void logInfo(String message)
void logInfo(int messageNum)
void logInfo(int messageNum, String param [,...])
void logInfo(int messageNum, Object[] paramArray)

void logWarning(String message)
void logWarning(int messageNum)
void logWarning(int messageNum, String param [,...])
void logWarning(int messageNum, Object[] paramArray)
```

Parameters

message The message text.

messageNum The number of a message in a message text file.

param A single parameter. There can be up to five parameters, separated by commas. Each is sequentially resolved to a parameter in the message text.

paramArray An array of parameters.

Return values

None.

Exceptions

None.

Notes

This method sends a message to the InterChange Server Express's logging destination. The logging destination can be a file, a window, or both.

By default, the logging destination is the file `InterchangeSystem.log`. You can change the logging destination by entering a value for the `LOG_FILE` parameter in the configuration file, `InterchangeSystem.cfg`. The parameter value can be a file name, `STDOUT` (which writes the log to the server's command window), or both.

Within each set of methods:

- The first form is self-contained and includes all of the text necessary to generate a message.
- The second form generates a message that does not have parameters.
- The third form contains a message number and a set of parameter values.
- The fourth form uses an array of parameters.

All forms of the method that take a *messageNum* parameter require the use of a message file that is indexed by message number. For information on how to set up a message text file, refer to Appendix A, “Message files,” on page 403.

Examples

The following example logs an informational message, using `getString()` to obtain an attribute value to log in the message.

```
logInfo("Item shipped. CustomerID: "
   + fromCustomerBusObj.getString("CustomerID"));
```

The following example logs an error message whose text is contained in the map message file. The message, which is number 10 in the message file, takes two parameters: customer last name (`LName` attribute) and customer first name (`FName` attribute).

```
logError(10, customer.get("LName"), customer.get("FName");
```

The following example logs an error message using an array of parameters. For the purpose of illustration, the example uses an array with just two parameters. The example declares the array `args`, which has two elements, the customer ID and the customer name. The `logError()` method then logs an error, using message number 12 and the values in the `args` array.

```
Object[] args = {
      fromCustomerBusObj.getString("CustomerID"),
      fromCustomerBusObj.getString("CustomerName");
}

logError(12, args);
```

See also

`trace()`

raiseException()

Raises an exception.

Syntax

```
void raiseException(String exceptionType, String message)

void raiseException(String exceptionType, int messageNum,
   String parameter[,...])

void raiseException(RunTimeEntityException exception)
```

Parameters

exceptionType One of the following IBM WebSphere InterChange Server Express-defined constants:

`AnyException`	Any type of exception
`AttributeException`	Attribute access problem. For example, the collaboration called `getDouble()` on a String attribute or called `getString()` on a nonexistent attribute.
`JavaException`	Problem with Java code that is not part of the IBM WebSphere InterChange Server Express API.
`ObjectException`	Business object passed to a method was invalid or a null object was accessed.
`OperationException`	Service call was improperly set up and could not be sent.
`ServiceCallException`	Service call failed. For example, a connector or application is unavailable.
`SystemException`	Any internal error within the IBM WebSphere InterChange Server Express system.

message
A text string that embeds the exception message in the method call.

messageNum
A reference to a numbered message in the map message file.

parameters
A value for the parameter in the message itself. There can be up to five parameters in the method call.

exception
The name of an exception object variable.

Return values

None.

Notes

The `raiseException()` method has three forms:

- The first form of the method creates a new exception, passing an exception type and a string. Use it to embed a message into the method call itself.
- The second form creates a new exception, passing an exception type and a reference to a message in the map message file. The method call can contain up to five parameters, separated with commas.
- The third form raises an exception object that the map has previously handled. For example, a transformation step might get an exception, assign it to a variable, and do some other work. Finally, the transformation step raises the exception.

Note: All forms of the method that take a *messageNum* parameter require the use of a message file that is indexed by message number. For information on how to set up a message text file, refer to Appendix A, "Message files," on page 403.

Examples

The following example uses the first form of the method to raise an exception of `ServiceCallException` type. The text is embedded in the method call.

```
raiseException(ServiceCallException,
   "Attempt to validate Customer failed.");
```

The next example raises an exception of `ServiceCallException` type. The message in the message file is as follows:

```
23
Customer update failed for CustomerID={1} CustomerName={2}
```

The `raiseException()` method invokes the message, retrieves the values of the message parameters from the `fromCustomer` variable, and passes them to the `raiseException()` call.

```
raiseException(ServiceCallException, 23,
   fromCustomer.getString("CustomerID"),
   fromCustomer.getString("CustomerName"));
```

The final example raises a previously handled exception. The system-defined variable `currentException` is an exception object that contains the exception.

```
raiseException(currentException);
```

releaseRelConnection()

Releases a connection to a relationship database.

Syntax

```
void releaseRelConnection(Boolean doCommit)
```

Parameters

doCommit	The flag that indicates whether this method should call the `DtpConnection.commit()` method before it releases the database connection.

Return values

None.

Exceptions

`DtpConnectionException` – If an error occurs while trying to release the database connection or if the requested commit or rollback has failed.

Notes

The `releaseRelConnection()` method releases the connection for this specific map. It commits or rolls back the database transactions based on the value of its *doCommit* argument, as follows:

- If *doCommit* is `true`, `releaseRelConnection()` assumes it was called after the successful completion of the operation on a database and therefore it is safe to commit the transaction.
- If *doCommit* is `false`, `releaseRelConnection()` assumes it was called as the result of an exception and therefore the transaction must be rolled back.

Once `releaseRelConnection()` has performed the chosen action on the database transaction, it releases the database connection that the current thread is exclusively using.

See also

`getRelConnection()`, `release()`

trace()

Generates a trace message.

Syntax

```
void trace(String traceMsg)
void trace(int traceLevel, String traceMsg)
void trace(int traceLevel, int messageNum)
void trace(int traceLevel, int messageNum, String param [,...])
void trace(int traceLevel, int messageNum, Object[] paramArray)
```

Parameters

traceLevel	The tracing level that causes the message to be generated.
traceMsg	A string that prints to the trace file.
messageNum	A number that represents a message in the map message file.
param	A single parameter. You can add additional single parameters, separated by commas, up to a total of five.
paramArray	An array of parameters.

Notes

The `trace()` method generates a message that the map prints if tracing is turned on. This method has five forms:

- The first form takes just a string message that appears when tracing is set to any level.
- The second form takes a trace level and a string message that appears when tracing is set to the specified level or a higher level.
- The third form takes a trace level and a number that represents a message in the map message file. The entire message text appears in the message file and is printed as it is, without parameters, when tracing is set to the specified level or a higher level.
- The fourth form takes a trace level, a number that represents a message in the map message file, and one or more parameters to be used in the message. You can send up to five parameter values to be used with the message by separating the values with commas.
- The fifth form takes a trace level, a number that represents a message in the map message file, and an array of parameter values.

Note: All forms of the method that take a *messageNum* parameter require the use of a message file that is indexed by message number. For information on how to set up a message text file, refer to Appendix A, “Message files,” on page 403.

You can set the trace level for a map as part of the Map Properties.

Examples

The following example generates a Level 2 trace message and supplies the text of the message:

```
trace (2, "Starting to trace at Level 2");
```

The following example prints message 201 in the map message file if the trace level is 2 or higher. The message has two parameters, a name and a year, for which this method call passes values.

```
trace(2, 201, "DAVID", "1961");
```

See also

logError(), logInfo(), logWarning()

Chapter 10. BusObj class

The methods documented in this chapter operate on objects of the BusObj class.

Note: The BusObj class is used for both collaboration development and mapping; check the Notes section for each method's usage issues.

The first two sections of this chapter explain the exceptions listed with these methods and how to specify attributes and child business objects in a hierarchical business object. The rest of the sections describe the methods listed in Table 92.

Table 92. BusObj method summary

Method	Description	Page
copy()	Copy all attribute values from the input business object to this one.	241
duplicate()	Create a business object (BusObj object) exactly like this one.	242
equalKeys()	Compare this business object's key attribute values with those in the input business object.	242
equals()	Compare this business object's attribute values with those in the input business object, including child business objects.	243
equalsShallow()	Compare this business object's attribute values with those in the input business object, excluding child business objects from the comparison.	244
exists()	Check for the existence of a business object attribute with a specified name.	244
getBoolean(), getDouble(), getFloat(), getInt(), getLong(), get(), getBusObj(), getBusObjArray(), getLongText(), getString()	Retrieve the value of a single attribute from a business object.	245
getLocale()	Retrieve the locale of the business object's data.	247
getType()	Retrieve the name of the business object definition on which this business object was based.	247
getVerb()	Retrieve this business object's verb.	247
isBlank()	Find out whether the value of an attribute is set to a zero-length string.	248
isKey()	Find out whether a business object's attribute is defined as a key attribute.	248
isNull()	Find out whether the value of a business object's attribute is null.	249
isRequired()	Find out whether a business object's attribute is defined as a required attribute.	250
keysToString()	Retrieve the values of a business object's primary key attributes as a string.	250
set()	Set a business object's attribute to a specified value of a particular data type.	251

© Copyright IBM Corp. 2003

Table 92. BusObj method summary (continued)

Method	Description	Page
setContent()	Set the contents of this business object to another business object.	252
setDefaultAttrValues()	Set all attributes to their default values.	253
setKeys()	Set the values of this business object's key attributes to the values of the key attributes in another business object.	253
setLocale()	Set the locale of the current business object.	253
setVerb()	Set the verb of a business object.	254
setWithCreate()	Set a business object's attribute to a specified value of a particular data type, creating an object for the value is one does not already exist.	254
toString()	Return the values of all attributes in a business object as a string.	255
validData()	Checks whether a specified value is a valid type for a specified attribute.	256

Exceptions and exception types

Methods for which exceptions or exception types are listed throw the `CollaborationException` exception. Some methods have both exceptions and exception types listed. Both of these relate to a `CollaborationException` object and differ as follows:

- An *Exception* is a class that is subclassed from `CollaborationException`. If there is a subclassed exception, you can use it in mapping to determine more closely the cause of the problem.
- An *Exception type* is a piece of data in a `CollaborationException` object. Collaboration developers use this exception type to catch exceptions through the Designer user interface. In addition, all users of `BusObj` can use this field to determine the reason for a failure if there is no exception class thrown that is more detailed than `CollaborationException`.

Syntax for traversing hierarchical business objects

When you are writing code that requires that you traverse hierarchical business objects, you need to use the syntax that lets you specify attributes in elements in child business object arrays that are elements of child business object arrays, and other such complexities. This chapter specifies the syntax to use.

An attribute specification can be:

```
[[attributeName[index].]...]attributeName
```

This syntax expands to any of the following formats:

```
attributeName
attributeName[index].attributeName
attributeName[index]... .attributeName
```

Note: Do not use the period (.) when creating a business object attribute name. If a business object attribute has a period within its name, a IBM WebSphere InterChange Server Express Map interprets the period as Java's dot operator

and imparts special meaning to it. For example, “attribute.name”will be interpreted as “name” being a field or method for the “attribute” object.

Specifying an attribute of basic type

The following example uses the `busObj.get()` method to retrieve a basic type attribute named `OrderID` from the business object `orderObj`.

```
orderObj.get("OrderID");
```

Specifying an attribute in a child business object

The following example assumes that `orderObj` is a hierarchical business object. One of its attributes is `CustomerInfo`, a single-cardinality child business object. The example retrieves the customer name from the `CustomerName` attribute of `CustomerInfo`.

```
orderObj.get("CustomerInfo.CustomerName");
```

Specifying an attribute in a child of a child business object

If there is a chain of child business objects, in which `CustomerInfo` is a child of `orderObj` and `AddressInfo` is a child of `CustomerInfo`, you can retrieve city information from `AddressInfo` as follows:

```
orderObj.get("CustomerInfo.AddressInfo.City");
```

Specifying an attribute in an element of an array of child business objects

You can also refer to a child business object in an array by specifying its index in the array. The first element in the array always begins with zero. For example, the following example retrieves the value of the Quantity attribute from the third child business object in an array.

```
orderObj.get("LineItem[2].Quantity");
```

copy()

Copy all attribute values from the input business object to this one.

Syntax

```
void copy(BusObj inputBusObj)
```

Parameters

inputBusObj The name of the business object whose attributes values are copied into the current business object.

Notes

The `copy()` method copies the entire business object, including all child business objects and child business object arrays. This method does not set a reference to the copied object. Instead, it clones all attributes; that is, it creates separate copies of the attributes.

Examples

The following example copies the values contained in `sourceCustomer` to `destCustomer`.

```
destCustomer.copy(sourceCustomer);
```

The following example creates three business objects (myBusObj, myBusObj2, and mysettingBusObj) and sets the attr1 attribute of myBusObj with the value in mysettingBusObj. It then clones all attributes of myBusObj to myBusObj2.

```
BusObj myBusObj = new BusObj();
BusObj myBusObj2 = new BusObj();

BusObj mySettingBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);
myBusObj2.copy(myBusObj);
```

After this code fragment executes, myBusObj.attr1 and myBusObj2.attr1 are *both* set to the mySettingBusObj business object. However, if mySettingBusObj is changed in any way, myBusObj.attr1 changes but myBusObj2.attr1 does not. Because the attributes of myBusObj2 were set with copy(), their values were cloned. Therefore, the value of attr1 in myBusObj2 is still the original mySettingBusObj.attr1 value *before* the change.

duplicate()

Create a business object (BusObj object) exactly like this one.

Syntax

```
BusObj duplicate()
```

Return values

The duplicate business object.

Exceptions

CollaborationException—The duplicate() method can set the following exception type for this exception: ObjectException.

Notes

This method makes a clone of the business object and returns it. You must explicitly assign the return value of this method call to a declared variable of BusObj type.

Examples

The following example duplicates sourceCustomer in order to create destCustomer.

```
BusObj destCustomer = sourceCustomer.duplicate();
```

equalKeys()

Compare this business object's key attribute values with those in the input business object.

Syntax

```
boolean equalKeys(BusObjinputBusObj)
```

Parameters

inputBusObj A business object to compare with this business object.

Return values

Returns `true` if the values of all key attributes are the same; returns `false` if they are not the same.

Exceptions

`CollaborationException`—The `equalKeys()` method can set the following exception type for this exception:

- `ObjectException` – Set if the business object argument is invalid.

See also

`equalsShallow()`, `equals()`

Notes

This method performs a shallow comparison; that is, it does not compare the keys in child business objects.

Examples

The following example compares the key values of `order2` to those in `order1`.

```
boolean areEqual = order1.equalKeys(order2);
```

---

equals()

Compare this business object's attribute values with those in the input business object, including child business objects.

Syntax

```
-boolean equals(Object inputBusObj)
```

Parameters

inputBusObj A business object to compare with this business object.

Return values

Returns `true` if the values of all attributes are the same; otherwise, returns `false`.

Exceptions

`CollaborationException`—The `equals()` method can set the following exception type for this exception:

- `ObjectException` – Set if the business object argument is invalid.

Notes

This method compares this business object's attribute values with those in the input business object. If the business objects are hierarchical, the comparison includes all attributes in the child business objects.

Note: Passing in the business object as an `Object` ensures that this `equals()` method overrides the `Object.equals()` method.

In the comparison, a null value is considered equivalent to any value to which it is compared and does not prevent a return of `true`.

See also

`equalsShallow()`, `equalKeys()`

Examples

The following example compares all attributes of `order2` to all attributes of `order1` and assigns the result of the comparison to the variable `areEqual`. The comparison includes the attributes of child business objects, if any.

```
boolean areEqual = order1.equals(order2);
```

equalsShallow()

Compare this business object's attribute values with those in the input business object, excluding child business objects from the comparison.

Syntax

```
boolean equalsShallow(BusObj inputBusObj)
```

Parameters

inputBusObj	A business object to compare with this business object.

Return values

Returns `true` if the values of all attributes are the same; otherwise, returns `false`.

Exceptions

`CollaborationException`—The `equalsShallow()` method can set the following exception type for this exception:

- `ObjectException` – Set if the business object argument is invalid.

See also

`equals()`, `equalKeys()`

Examples

The following example compares attributes of `order2` with attributes of `order1`, excluding the attributes of child business objects, if any.

```
boolean areEqual = order1.equalsShallow(order2);
```

exists()

Check for the existence of a business object attribute with a specified name.

Syntax

```
boolean exists(String attribute)
```

Parameters

attribute	The name of an attribute.

Return values

Returns `true` if the attribute exists; otherwise, returns `false` if the attribute does not exist.

Examples

The following example checks whether business object `order` has an attribute called `Notes`.

```
boolean notesAreHere = order.exists("Notes");
```

getBoolean(), getDouble(), getFloat(), getInt(), getLong(), get(), getBusObj(), getBusObjArray(), getLongText(), getString()

Retrieve the value of a single attribute from a business object.

Syntax

```
Object get(String attribute)
Object get(int position)
```

```
boolean getBoolean(String attribute)
double getDouble(String attribute)
float getFloat(String attribute)
int getInt(String attribute)
long getLong(String attribute)
Object get(String attribute)
BusObj getBusObj(String attribute)
BusObjArray getBusObjArray(String attribute)
String getLongText(String attribute)
String getString(String attribute)
```

Parameters

attribute The name of an attribute.

position an integer that specifies the ordinal position of an attribute in the business object's attribute list.

Return values

The value of the specified attribute.

Exceptions

`CollaborationException`—These `get` methods can set the following exception type for this exception:

- `AttributeException` - Set if an attribute access problem occurs. For example, this exception can be caused if the collaboration calls `getDouble()` on a String attribute that does not consist of digits or calls `getString()` on a nonexistent attribute.

Notes

The `get()` method retrieves an attribute value from the current business object. It returns a copy of the attribute value. It does *not* return an object reference to this attribute in the source business object. Therefore, any change to attribute value in the source business object is *not* made to the value that `get()` returns. Each time this method is called, it returns a new copy (clone) of the attribute.

The `get()` method provides the following forms:

- The first form returns a value of the type specified in the method name. For example, `getBoolean()` returns a boolean value, `getBusObj()` returns a BusObj value, `getDouble()` returns a double value, and so on. However, `getLongText()` returns a String object because the WebSphere InterChange Server Express

longtext type is a String object with no maximum size. Use these forms to retrieve attributes with specific basic or WebSphere InterChange Server Express-defined data types.

These methods provide the ability to access an attribute value by specifying the *name* of the attribute.

- The second form, `get()` retrieves the value of an attribute of *any* type. You can cast the returned value to the appropriate value of the attribute type.

  This method provides the ability to access an attribute value by specifying *either* the *name* of the attribute or the attribute's index *position* within the business object attribute list.

Examples

The following example illustrates how `get()` returns a copy (clone) of the attribute value instead of an object reference:

```
BusObj mySettingBusObj = new BusObj();
BusObj myBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);

BusObj Extract = myBusObj.get("attr1");
```

After this code fragment executes, if you change the `Extract` business object, `mySettingBusObj` does *not* change because the `get()` call returned a copy of the `attr1` attribute.

The following example uses `getBusObj()` to retrieve a child business object containing a customer address from the `customer` business object and assign it to the variable `address`.

```
BusObj address = customer.getBusObj("Address");
```

The following example uses `getString()` to retrieve the value of the `CustomerName` attribute. The business object variable is `sourceCustomer`.

```
String customerName = sourceCustomer.getString("CustomerName");
```

The following example uses `getInt()` to retrieve the `Quantity` values from two business objects whose variables are `item1` and `item2`. The example then computes the sum of both quantities.

```
int sumQuantity = item1.getInt("Quantity") + item2.getInt("Quantity");
```

The following example retrieves the attribute `Item` from the business object variable `order`. The attribute `Item` is a business object array.

```
BusObjArray items = order.getBusObjArray("Item");
```

The following example gets the `CustID` attribute value from the source business object and sets the `Customer` value in the destination business object to match.

```
destination.set("Customer", source.get("CustID"));
```

The following example accesses an attribute value using the attribute's ordinal position within the attribute list:

```
for i=0; i<maxAttrCount; i++)
{
       String strValue = (String)myBusObj.get(i);
    ...
```

getLocale()

Retrieve the locale associated with the business object's data.

Syntax

```
java.util.Locale getLocale()
```

Parameters

None.

Return values

A Java Locale object that contains information about the business object's locale. This Locale object must be an instance of the `java.util.Locale` class.

Notes

The `getLocale()` method returns the locale associated with the data in a business object. This locale is often different from the collaboration locale in which the collaboration is executing.

See also

getLocale() (`BaseCollaboration` class), setLocale()

getType()

Retrieve the name of the business object definition on which this business object was based.

Syntax

```
String getType()
```

Return values

The name of a business object definition.

Notes

The type of a business object, in terms of this method, is the name of the business object definition from which the business object was created.

Returns

The following example retrieves the type of a business object called `sourceShipTo`.

```
String typeName = sourceShipTo.getType();
```

The following example copies a triggering event into a new business object of the appropriate type.

```
BusObj source = new BusObj(triggeringBusObj.getType());
```

getVerb()

Retrieve this business object's verb.

Syntax

```
String getVerb()
```

Return values

The name of a verb, such as Create, Retrieve, Update, or Delete.

Notes

In collaboration development, this method is useful for scenarios that handle multiple types of incoming events. The first action node in a scenario calls `getVerb()`. The outgoing transition links from that action node then test the contents of the returned string, so that each outgoing transition link is the start of an execution path that handles one of the possible verbs.

Examples

The following example obtains the verb from a business object called `orderEvent` and assigns it to a variable called `orderVerb`.

```
String orderVerb = orderEvent.getVerb();
```

isBlank()

Find out whether the value of an attribute is set to a zero-length string.

Syntax

```
boolean isBlank(String attribute)
```

Parameters

attribute The name of an attribute.

Returns

Returns `true` if the attribute value is a zero-length string; returns `false` otherwise.

Notes

A zero-length string can be compared to the string "". It is different from a null, whose presence is detected by the `isNull()` method.

If a collaboration needs to retrieve an attribute value and then do something with it, it can call `isBlank()` and `isNull()` to check that it has a value before retrieving the value.

Examples

The following example checks whether the `Material` attribute of the `sourcePaperClip` business object is a zero-length string.

```
boolean key = sourcePaperClip.isBlank("Material");
```

isKey()

Find out whether a business object's attribute is defined as a key attribute.

Syntax

```
boolean isKey(String attribute)
```

Parameters

attribute The name of an attribute.

Return values

Returns `true` if the attribute is a key attribute; returns `false` if it is not a key attribute.

Examples

The following example determines whether the `CustID` attribute of the `customer` business object is a key attribute.

```
boolean keyAttr = (customer.isKey("CustID"));
```

---

isNull()

Find out whether the value of a business object's attribute is null.

Syntax

```
boolean isNull(String attribute)
```

Parameters

attribute The name of an attribute.

Return values

Returns `true` if the attribute value is null; returns `false` if it is not null.

Notes

A null indicates no value, in contrast to a zero-length string value, which is detected by calling `isBlank()`. Test an object with `isNull()` before using it, because if the object is null, the operation could fail.

An attribute value can be null under these circumstances:

- The attribute value was explicitly set to null.

  An attribute value can be set to null using the `set()` method.
- The attribute value was never set.

  At instantiation of a new business objects, all attribute values are initialized with a null. If the attribute value has not been set between the time of creation and the time of the `isNull()` call, the value is still null.
- The null was inserted during mapping.

  When a collaboration is processing a business object received from a connector, the mapping process might have inserted the null. The mapping process converts the application-specific business object received from the connector to the generic business object handled by the collaboration. For each attribute in the generic business object that has no equivalent in the application-specific object, the map inserts a null value.

Tip: Always call `isNull()` before performing an operation on an attribute that is a child business object or child business object array, because Java does not allow operations on null objects.

Examples

The following example checks whether the `Material` attribute of the `sourcePaperClip` business object has a null value.

```
boolean key = sourcePaperClip.isNull("Material");
```

The following example checks whether the `CustAddr` attribute of the `contract1` business object is null before retrieving it. The attribute retrieval proceeds only if the `isNull()` check is false, showing that the attribute is not null.

```
if (! contract1.isNull("CustAddr"))
   {
   BusObj customerAddress = contract1.getBusObj("CustAddr");
   //do something with the "customerAddress" business object
   }
```

isRequired()

Find out whether a business object's attribute is defined as a required attribute.

Syntax

```
boolean isRequired(String attribute)
```

Parameters

attribute The name of an attribute.

Return values

Returns `true` if the attribute is required; returns `false` if it is not required.

Notes

If an attribute is defined as required, it must have a value and the value must not be a null.

Examples

The following example logs a warning if a required attribute has a null value.

```
if ( (customer.isRequired("Address"))
      && (customerBusObj.isNull("Address)) )
   {
   logWarning(12, "Address is required and cannot be null.");
   }
else
   {
   //do something else
   }
```

keysToString()

Retrieve the values of a business object's primary key attributes as a string.

Syntax

```
String keysToString()
```

Return values

A String object containing all the key values in a business object, concatenated, and ordered by the ordinal value of the attributes.

Notes

The output from this method contains the name of the attribute and its value. Multiple values are primary key attribute values, concatenated and separated by spaces. For example, if there is one primary key attribute, SS#, this could be the output:

```
SS#=100408394
```

If the primary key attributes are FirstName and LastName, this could be the output:

```
FirstName=Nina LastName=Silk
```

Examples

The following example returns the values of key attributes of the business object represented by the variable name `fromOrder`.

```
String keyValues = fromOrder.keysToString();
```

set()

Set a business object's attribute to a specified value of a particular data type.

Syntax

```
void set(String attribute, Object value)
void set(int position, Object value)
```

```
void set(String attribute, boolean value)
void set(String attribute, double value)
void set(String attribute, float value)
void set(String attribute, int value)
void set(String attribute, long value)
void set(String attribute, Object value)
void set(String attribute, String value)
```

Parameters

attribute The name of the attribute to set.

position An integer that specifies the ordinal position of an attribute in the business object's attribute list.

value An attribute value.

Exceptions

`CollaborationException`—The `set()` method can set the following exception type for this exception:

- `AttributeException`—Set if an attribute access problem occurs.

Notes

The `set()` method sets an attribute value in the current business object. This method sets an object reference to the *value* parameter when it assigns the value to the attribute. It does *not* clone the attribute value from the source business object. Therefore, any changes to *value* in the source business object are also made to the attribute in the business object that calls `set()`.

The `set()` method provides the following forms:

- The first form sets a value of the type specified by the method's second parameter type. For example, set(String *attribute*, boolean *value*) sets an attribute

with a boolean value, set(String *attribute*, double *value*) sets an attribute with a double value, and so on. Use this form to set attributes with specific basic or WebSphere InterChange Server Express-defined data types.

These methods provide the ability to access an attribute value by specifying the *name* of the attribute.

- The second form sets the value of an attribute of *any* type. You can send in any data type as the attribute value because the attribute-value parameter is of type Object. For example, to set an attribute that is of `BusObj` or `LongText` object, use this form of the method and pass in the `BusObj` or `LongText` object as the attribute value.

  This form of the set() method provides the ability to access an attribute value by specifying *either* the *name* of the attribute or the attribute's index *position* within the business object attribute list.

Examples

The following example sets the `LName` attribute in `toCustomer` to the value `Smith`.

```
toCustomer.set("LName", "Smith");
```

The following example illustrates how `set()` assigns an object reference instead of cloning the value:

```
BusObj BusObj myBusObj = new BusObj();
BusObj mySettingBusObj = new BusObj();

myBusObj.set("attr1", mySettingBusObj);
```

After this code fragment executes, the `attr1` attribute of `myBusObj` is set to the `mySettingBusObj` business object. If `mySettingBusObj` is changed in any way, `myBusObj.attr1` is changed in the exact manner because `set()` makes an object reference to `mySettingBusObj` when it sets the `attr1` attribute; it does *not* create a static copy of `mySettingBusObj`.

The following example sets an attribute value using the attribute's ordinal position within the attribute list:

```
for i=0; i<maxAttrCount; i++)
{
       myBusObj.set(i, strValue);
    ...
```

setContent()

Set the contents of this business object to another business object.

Syntax

```
void setContent(BusObj BusObj)
```

Parameters

BusObj The business object whose values are used to set values of this business object.

Exceptions

`CollaborationException`—The `setContent()` method can set one of the following exception types for this exception:

- `AttributeException` – Set if an attribute access problem occurs.

- `ObjectException` – Set if the business object argument is invalid.

Examples

The following example sets the contents of the instance variable for the output object `ObjOutput1` to the contents of the business `object rDstBO[0]`.

```
ObjOutput1.setContent(rDstBO[0]);
```

setDefaultAttrValues()

Set all attributes to their default values.

Syntax

```
void setDefaultAttrValues()
```

Notes

A business object definition can include default values for attributes. The method sets the values of this business object's attributes to the values specified as defaults in the definition.

Examples

The following example sets the values of the `PaperClip` business object to their default values:

```
PaperClip.setDefaultAttrValues();
```

setKeys()

Set the values of this business object's key attributes to the values of the key attributes in another business object.

Syntax

```
void setKeys(BusObj inputBusObj)
```

Parameters

inputBusObj	The business object whose values are used to set values of another business object

Exceptions

`CollaborationException`—The `setKeys()` method can set one of the following exception types for this exception:

- `AttributeException` – Set if an attribute access problem occurs.
- `ObjectException` – Set if the business object argument is invalid.

Examples

The following example sets the key values in the business object `helpdeskCustomer` to the key values in the business object `ERPCustomer`.

```
helpdeskCustomer.setKeys(ERPCustomer);
```

setLocale()

Set the locale of the current business object.

Syntax

```
void setLocale(java.util.Locale locale
```

Parameters

locale The Java Locale object that contains the information about the locale to assign to the business object. This Locale object must be an instance of the `java.util.Locale` class.

Return values

None.

Notes

The `setLocale()` method assigns a locale to the data associated with a business object. The locale might be different from the collaboration locale in which the collaboration executes.

See also

getLocale()

setVerb()

Set the verb of a business object.

Syntax

```
void setVerb(String verb)
```

Parameters

verb The verb of the business object.

Notes

The `setVerb()` method is used only in mapping.

Note: Do *not* use this method in collaboration development, where you must set the verb of an outgoing business object interactively by filling in the properties of a service call.

Examples

The following example sets the verb Delete on the business object contactAddress.

```
contactAddress.setVerb("Delete");
```

setWithCreate()

Set a business object's attribute to a specified value of a particular data type, creating an object for the value is one does not already exist.

Syntax

```
void setWithCreate(String attributeName, BusObj busObj)
void setWithCreate(String attributeName, BusObjArray busObjArray)
void setWithCreate(String attributeName, Object value)
```

Parameters

attributeName	The name of the attribute to set.
busObj	The business object to insert into the target attribute.
busObjArray	The business object array to insert into the target attribute.
value	The object to insert into the target attribute. This object needs to be one of the following types: `BusObj`, `BusObjArray`, `Object`.

Exceptions

`CollaborationException`—The `setWithCreate()` method can set the following exception type for this exception:

- `AttributeException`—Set if an attribute access problem occurs.

Notes

If the object provided is a `BusObj` and the target attribute contains multi-cardinality child business object, the `BusObj` is appended to the `BusObjArray` as its last element. If the target attribute contains a `BusObj`, however, this business object replaces the previous value.

Examples

The following example sets an attribute called `ChildAttrAttr` to the value 5. The attribute is found in a business object contained in myBO's attribute, `ChildAttr`. If the `childAttr` business object does not exist at the time of the call, this method call creates it.

```
myBO.setWithCreate("childAttr.childAttrAttr", "5");
```

toString()

Return the values of all attributes in a business object as a string.

Syntax

```
String toString()
```

Return values

A `String` object containing all attribute values in a business object.

Notes

The string that results from a call to this method is similar to the following example:

```
Name: GenEmployee
Verb: Create
Type: AfterImage
Attributes: (Name, Type, Value)

LastName:String, Davis
FirstName:String, Miles
SS#:String, 041-33-8989
Salary:Float, 15.00
ObjectEventId:String, MyConnector_922323619411_1
```

Examples

The following example returns a string containing the attribute values of the business object variable `fromOrder`.

```
String values = fromOrder.toString();
```

validData()

Checks whether a specified value is a valid type for a specified attribute.

Syntax

```
boolean validData(String attributeName, Object value)
boolean validData(String attributeName, BusObj value)
boolean validData(String attributeName, BusObjArray value)
boolean validData(String attributeName, String value)
boolean validData(String attributeName, long value)
boolean validData(String attributeName, int value)
boolean validData(String attributeName, double value)
boolean validData(String attributeName, float value)
boolean validData(String attributeName, boolean value)
```

Parameters

attributeName The attribute.

value The value.

Returns

true or false (boolean return)

Notes

Checks the compatibility of the value passed in with the target attribute (as specified by *attributeName*). These are the criteria:

for primitive types (String, long, int, double, float, boolean)	the value must be convertible to the data type of the attribute
for a `BusObj`	the value must have the same type as that of the target attribute
for a `BusObjArray`	the value must point to a `BusObj` or `BusObjArray` with the same (business object definition) type as that of the attribute
for an `Object`	the value must be of type `String`, `BusObj`, or `BusObjArray`. The corresponding validation rules are then applied.

Deprecated methods

Some methods in the `BusObj` class were supported in earlier versions but are no longer supported. These *deprecated methods* will not generate errors, but CrossWorlds recommends that you avoid their use and migrate existing code to the new methods. The deprecated methods might be removed in a future release.

Table 93 lists the deprecated methods for the `BusObj` class. If you have not used Map Designer Express before, ignore this section.

Table 93. Deprecated methods, BusObj Class

Former Method	Replacement
`getCount()`	`BusObjArray.size()`
`getKeys()`	`keysToString()`
`getValues()`	`toString()`
`not`	standard Java NOT operator, `"!"`
`set(BusObj` *`inputBusObj`*`)`	`copy()`
All methods that took a child business object or child business object array as an input argument	Get a handle to the child business object or business object array and use the methods of the `BusObj` or `BusObjArray` class

The `setVerb()` method, which was previously listed as deprecated, is now restored for use in mapping. Do not use it within a collaboration.

Chapter 11. BusObjArray class

The methods documented in this chapter operate on objects of the IBM WebSphere InterChange Server Express-defined class `BusObjArray`. The `BusObjArray` class encapsulates an array of business objects. In a hierarchical business object, an attribute is a reference to an array of child business objects when its cardinality is equal to `n`. Operations on the `BusObjArray` class can return either a `BusObjArray` object or an actual array of business objects.

Note: The `BusObjArray` class is used for both collaboration development and mapping; check the Notes section for each method's usage issues.

Table 94 lists the methods of the `BusObjArray` class.

Table 94. BusObjArray method summary

Method	Description	Page
`addElement()`	Add a business object to this business object array.	260
`duplicate()`	Create a business object array (`BusObjArray` object) exactly like this one.	260
`elementAt()`	Retrieve a single business object by specifying its position in this business object array.	261
`equals()`	Compare another business object array with this one.	261
`getElements()`	Retrieve the contents of this business object array.	262
`getLastIndex()`	Retrieve the last available index from a business object array.	262
`max()`	Retrieve the maximum value for the specified attribute among all elements in this business object array.	262
`maxBusObjArray()`	Returns the business objects that have the maximum value for the specified attribute, as a business object array (`BusObjArray` object).	263
`maxBusObjs()`	Returns the business objects that have the maximum value for the specified attribute, as an array of `BusObj` objects.	264
`min()`	Retrieve the minimum value for the specified attribute among the business objects in this array.	265
`minBusObjArray()`	Returns the business objects that have the minimum value for the specified attribute, as a `BusObjArray` object.	266
`minBusObjs()`	Returns the business objects that have the minimum value for the specified attribute, as an array of `BusObj` objects.	267
`removeAllElements()`	Remove all elements from this business object array.	268
`removeElement()`	Remove a business object element from a business object array.	268
`removeElementAt()`	Remove an element at a particular position in this business object array.	269

© Copyright IBM Corp. 2003

Table 94. BusObjArray method summary (continued)

Method	Description	Page
`setElementAt()`	Set the value of a business object in a business object array.	269
`size()`	Return the number of elements in this business object array.	270
`sum()`	Adds the values of the specified attribute for all business objects in this business object array.	270
`swap()`	Reverse the positions of two business objects in this business object array. Keep in mind that the first element in the array is zero (0), the second is 1, the third is 2, and so on.	270
`toString()`	Retrieve the values in this business object array as a single string.	271

Note: See "Exceptions and exception types" on page 240 for an important clarification on exception handling with this class. The section applies to exceptions in `BusObjArray` and `BusObj` only.

addElement()

Add a business object to this business object array.

Syntax

```
void addElement(BusObj element)
```

Parameters

element A business object to add to the array.

Exceptions

`CollaborationException`—The `addElement()` method can set the following exception type for this exception:

- `AttributeException` – Set if the element is not valid.

Examples

The following example uses the `getBusObjArray()` method to retrieve an array of business objects called `itemList` from the business object `order`. The array is assigned to `items`, and then a new business object is added to `items`.

```
BusObjArray items = order.getBusObjArray("itemList");
items.addElement(new BusObj("oneItem"));
```

duplicate()

Create a business object array (`BusObjArray` object) exactly like this one.

Syntax

```
BusObjArray duplicate()
```

Return values

A business object array.

Examples

The following example duplicates the `items` array, creating `newItems`.

```
BusObjArray newItems = items.duplicate();
```

elementAt()

Retrieve a single business object by specifying its position in this business object array.

Syntax

```
BusObj elementAt(int index)
```

Parameters

index The array element to retrieve. The first element in the array is zero (0), the second is 1, the third is 2, and so on.

Exceptions

`CollaborationException`—The `elementAt()` method can set the following exception type for this exception:

- `AttributeException` – Set if the element is not valid.

Examples

The following example retrieves the 11th business object in the `items` array and assigns it to the `Item` variable.

```
BusObj Item = items.elementAt(10);
```

equals()

Compare another business object array with this one.

Syntax

```
boolean equals(BusObjArray inputBusObjArray)
```

Parameters

inputBusObjArray
A business object array to compare with this business object array.

Notes

The comparison between the two business object arrays checks the number of elements and their attribute values.

Examples

The following example uses `equals()` to set up a conditional loop, the inside of which is not shown.

```
if (items.equals(newItems))
   {
...
   }
```

getElements()

Retrieve the contents of this business object array.

Syntax

```
BusObj[] getElements()
```

Exceptions

`CollaborationException`—The `getElements()` method can set the following exception type for this exception:

- `ObjectException` – Set if one of the elements is not valid.

Examples

The following example prints the elements of the items array.

```
BusObj[] elements = items.getElements();
for (int i=0, i<elements.length; i++)
   {
   trace(1, elements[i].toString());
   }
```

getLastIndex()

Retrieve the last available index from a business object array.

Syntax

```
int getLastIndex()
```

Returns

The last index to the last element in this `BusObjArray`.

Notes

Previously, the `size()` method was used to do this. That is, the user would use the `size()` of the business object array to retrieve the last index available in a BusObjArray. Unfortunately, this approach yields incorrect data if the BusObjArray contains gaps.

Like all Java arrays, `BusObjArray` is a zero relative array. This means that the `size()` method will return 1 greater than the `getLastIndex()` method.

Examples

The following example retrieves the last index in the business object array.

```
int lastElementIndex = items.getLastIndex();
```

max()

Retrieve the maximum value for the specified attribute among all elements in this business object array.

Syntax

```
String max(String attr)
```

Parameters

attr A variable that refers to an attribute in the business object. The attribute must be one of these types: `String`, `LongText`, `Integer`, `Float`, and `Double`.

Returns

The maximum value of the specified attribute in the form of a string, or null if the value for that attribute is null for all elements in this `BusObjArray`.

Exceptions

`UnknownAttributeException` – When the specified attribute is not a valid attribute in the business objects passed in.

`UnsupportedAttributeTypeException` – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from `CollaborationException`. The `max()` method can set the following exception type for these exceptions: `AttributeException`.

Notes

The `max()` method looks for the maximum value for the specified attribute among the business objects in this `BusObjArray`. For example, if three employee objects are used, and the attribute is "Salary" which is of type "Float," it will return the string representing the largest salary.

If the value of the specified attribute for an element in `BusObjArray` is null, then that element is ignored. If the value of the specified attribute is null for all elements, then null is returned.

When the attribute type is of type String, `max()` returns the attribute value that is the longest string lexically.

Examples

```
String maxSalary = items.max("Salary");
```

maxBusObjArray()

Returns the business objects that have the maximum value for the specified attribute, as a business object array (`BusObjArray` object).

Syntax

```
BusObjArray maxBusObjArray(String attr)
```

Parameters

attr A `String`, `LongText`, `Integer`, `Float`, or `Double` variable that refers to an attribute in a business object in the business object array.

Returns

A list of business objects in the form of `BusObjArray` or `null`.

Exceptions

UnknownAttributeException – When the specified attribute is not a valid attribute in the business objects passed in.

UnsupportedAttributeTypeException – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from CollaborationException. The maxBusObjArray() method can set the following exception type for these exceptions: AttributeException.

Notes

The maxBusObjArray() method finds one or more business objects with the maximum value for the specified attribute, and returns these business objects in a BusObjArray object.

For example, suppose that this is a business object array containing Employee business objects and that the input argument is the attribute Salary, a Float. The method determines the largest value for Salary in all the Employee business objects and returns the business object that contains that value. If multiple business objects have that largest Salary value, the method returns all of those business objects.

A business object is ignored if the specified attribute contains null. If the value is null in all business objects in the array, null is returned.

When the attribute is of type String, the method returns the longest string lexically.

Examples

```
BusObjArray boarrayWithMaxSalary = items.maxBusObjArray("Salary");
```

maxBusObjs()

Returns the business objects that have the maximum value for the specified attribute, as an array of BusObj objects.

Syntax

```
BusObj[] maxBusObjs(String attr)
```

Parameters

attr	A String, LongText, Integer, Float, or Double variable that refers to an attribute in the business object.

Returns

A list of business objects in the form of a BusObj[] or null.

Exceptions

UnknownAttributeException – When the specified attribute is not a valid attribute in the business objects passed in.

UnsupportedAttributeTypeException – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from CollaborationException. The maxBusObjs() method can set the following exception type for these exceptions: AttributeException.

Notes

The maxBusObjs() method finds one or more business objects with the maximum value for the specified attribute, and returns these business objects as an array of BusObj objects.

For example, suppose that this is a business object array containing Employee business objects and that the input argument is the attribute Salary, a Float. The method determines the largest value for Salary in all the Employee business objects and returns the business object that contains that value. If multiple business objects have that largest Salary value, the method returns all of those business objects.

A business object is ignored if the specified attribute contains null. If the value is null in all business objects in the array, null is returned.

When the attribute is of type String, the method returns the longest string lexically.

Examples

```
BusObj[] bosWithMaxSalary = items.maxBusObjs("Salary");
```

---

min()

Retrieve the minimum value for the specified attribute among the business objects in this array.

Syntax

```
String min(String attr)
```

Parameters

attr	A String, LongText, Integer, Float, or Double variable that refers to an attribute in the business object.

Returns

The minimum value of the specified attribute in the form of a string, or null if the value for that attribute is null for all elements in this BusObjArray.

Exceptions

UnknownAttributeException – When the specified attribute is not a valid attribute in the business objects passed in.

UnsupportedAttributeTypeException – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from CollaborationException. The min() method can set the following exception type for these exceptions: AttributeException.

Notes

The `min()` method looks for the minimum value for the specified attribute among the business objects in this business object array.

For example, suppose that this is a business object array containing Employee business objects and that the input argument is the attribute Salary, a Float. The method determines the smallest value for Salary in all the Employee business objects and returns the business object that contains that value. If multiple business objects have that lowest Salary value, the method returns all of those business objects.

A business object is ignored if the specified attribute contains null. If the value is null in all business objects in the array, null is returned.

When the attribute is of type String, the method returns the shortest string lexically.

Examples

```
String minSalary = items.min("Salary");
```

minBusObjArray()

Returns the business objects that have the minimum value for the specified attribute, as a `BusObjArray` object.

Syntax

```
BusObjArray minBusObjArray(String attr)
```

Parameters

attr A `String`, `LongText`, `Integer`, `Float`, or `Double` variable that refers to an attribute in the business object.

Returns

A list of business objects in the form of `BusObjArray` or `null`.

Exceptions

`UnknownAttributeException` – When the specified attribute is not a valid attribute in the business objects passed in.

`UnsupportedAttributeTypeException` – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from `CollaborationException`. The `minBusObjArray()` method can set the following exception type for these exceptions: `AttributeException`.

Notes

The `minBusObjArray()` method finds one or more business objects with the minimum value for the specified attribute, and returns these business objects in a `BusObjArray` object.

For example, suppose that this is a business object array containing Employee business objects and that the input argument is the attribute Salary, a Float. The method determines the smallest value for Salary in all the Employee business objects and returns the business object that contains that value. If multiple business objects have that smallest Salary value, the method returns all of those business objects.

A business object is ignored if the specified attribute contains null. If the value is null in all business objects in the array, null is returned.

When the attribute is of type String, the method returns the shortest string lexically.

Examples

```
BusObjArray boarrayWithMinSalary = items.minBusObjArray("Salary");
```

---

minBusObjs()

Returns the business objects that have the minimum value for the specified attribute, as an array of `BusObj` objects.

Syntax

```
BusObj[] minBusObjs(String attr)
```

Parameters

attr	A `String`, `LongText`, `Integer`, `Float`, or `Double` variable that refers to an attribute in the business object.

Returns

A list of business objects in the form of a `BusObj[]` or `null`.

Exceptions

`UnknownAttributeException` – When the specified attribute is not a valid attribute in the business objects passed in.

`UnsupportedAttributeTypeException` – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from `CollaborationException`. The `minBusObjs()` method can set the following exception type for these exceptions: `AttributeException`.

Notes

The `minBusObjs()` method finds one or more business objects with the maximum value for the specified attribute, and returns these business objects as an array of `BusObj` objects.

For example, suppose that this is a business object array containing Employee business objects and that the input argument is the attribute Salary, a Float. The method determines the smallest value for Salary in all the Employee business

objects and returns the business object that contains that value. If multiple business objects have that smallest Salary value, the method returns all of those business objects.

A business object is ignored if the specified attribute contains null. If the value is null in all business objects in the array, null is returned.

When the attribute is of type String, the method returns the shortest string lexically.

Examples

```
BusObj[] bosWithMinSalary = items.minBusObjs("Salary");
```

removeAllElements()

Remove all elements from this business object array.

Syntax

```
void removeAllElements()
```

Examples

The following example removes all elements of the array `items`.

```
items.removeAllElements();
```

removeElement()

Remove a business object element from a business object array.

Syntax

```
void removeElement(BusObj element)
```

Parameters

elementReference
A variable that refers to an element of the array.

Exceptions

`CollaborationException`—The `removeElement()` method can set the following exception type for this exception:

- `AttributeException` – Set if the element is not valid.

Notes

After you delete an element from the array, the array resizes, changing the indexes of existing elements.

Examples

The following example deletes the element `Child1` from the business object array `items`.

```
items.removeElement(Child1);
```

removeElementAt()

Remove an element at a particular position in this business object array.

Syntax

```
void removeElementAt(int index)
```

Notes

After an element is removed from the array, the array resizes, possibly changing the indexes of existing elements.

Parameters

index The element index.

Exceptions

`CollaborationException`—The `removeElementAt()` method can set the following exception type for this exception:

- `AttributeException` – Set if the element is not valid.

Examples

The following example deletes the sixth business object in the array `items`.

```
items.removeElementAt(5);
```

setElementAt()

Set the value of a business object in a business object array.

Syntax

```
void setElementAt (int index, BusObj element)
```

Parameters

index An integer representing the array position. The first element in the array is zero (0), the second is 1, the third is 2, and so on.

inputBusObj The business object containing the values to which you want to set the array element.

Exceptions

`CollaborationException`—The `setElementAt()` method can set the following exception type for this exception:

- `AttributeException` – Set if the element is not valid.

Notes

This method sets the values of the business object at a specified array position to the values of an input business object.

Examples

The following example creates a new business object of type Item and adds it to the array `items`, as the fourth element.

```
items.setElementAt(5, new BusObj("Item"));
```

size()

Return the number of elements in this business object array.

Syntax

```
int size()
```

Notes

Like all Java arrays, BusObjArray is a zero relative array. This means that the `size()` method will return 1 greater than the `getLastIndex()` method.

Examples

The following example returns the number of elements in the array `items`.

```
int size = items.size();
```

sum()

Adds the values of the specified attribute for all business objects in this business object array.

Syntax

```
double sum(String attrName)
```

Parameters

attr	A variable that refers to an attribute in the business object. The attribute must be of type Integer, Float, or Double.

Returns

The sum of the specified attribute from the list of the business objects.

Exceptions

`UnknownAttributeException` – When the specified attribute is not a valid attribute in the business objects passed in.

`UnsupportedAttributeTypeException` – When the type of the specified attribute is not one of the supported attribute types listed in the note section.

All of the above exceptions are subclassed from `CollaborationException`. The `sum()` method can set the following exception type for these exceptions: `AttributeException`.

Examples

```
double sumSalary = items.sum("Salary");
```

swap()

Reverse the positions of two business objects in this business object array. Keep in mind that the first element in the array is zero (0), the second is 1, the third is 2, and so on.

Syntax

```
void swap(int index1, int index2)
```

Parameters

index1 The array position of one element you want to swap.

index2 The array position of the other element you want to swap.

Examples

The following example uses `swap()` to reverse the positions of BusObjA and BusObjC in the following array:

BusObjA	BusObjB	BusObjC

```
swap(0,2);
```

The result of the swap() call is the following array:

BusObjC	BusObjB	BusObjA

toString()

Retrieve the values in this business object array as a single string.

Syntax

```
String toString()
```

Examples

The following example uses `toString()` to retrieve the contents of the `items` business object array and then uses `logInfo()` to write the contents to the log file.

```
logInfo(items.toString());
```

Chapter 12. CwDBConnection class

The `CwDBConnection` class provides methods for executing SQL queries in a database. Queries are performed through a connection, which is obtained from a connection pool. To instantiate this class, you must call `getDBConnection()` in the `BaseDLM` class. All maps are derived or subclassed from `BaseDLM` so they have access to `getDBConnection()`.

Table 95 summarizes the methods in the `CwDBConnection` class.

Table 95. CwDBConnection method summary

Method	Description	Page
`beginTransaction()`	Begins an explicit transaction for the current connection.	273
`commit()`	Commits the active transaction associated with the current connection.	274
`executeSQL()`	Executes a static SQL query by specifying its syntax and an optional parameter array.	276
`executePreparedSQL()`	Executes a prepared SQL query by specifying its syntax and an optional parameter array.	275
`executeStoredProcedure()`	Executes an SQL stored procedure by specifying its name and parameter array.	278
`getUpdateCount()`	Returns the number of rows affected by the last write operation to the database.	279
`hasMoreRows()`	Determines whether the query result has more rows to process.	279
`inTransaction()`	Determines whether a transaction is in progress in the current connection.	280
`isActive()`	Determines whether the current connection is active.	280
`nextRow()`	Retrieves the next row from the query result.	281
`release()`	Releases use of the current connection, returning it to its connection pool.	281
`rollBack()`	Rolls back the active transaction associated with the current connection.	282

beginTransaction()

Begins an explicit transaction for the current connection.

Syntax

```
void beginTransaction()
```

Parameters

None.

Return values

None.

Exceptions

`CwDBConnectionException` – If a database error occurs.

© Copyright IBM Corp. 2003

Notes

The `beginTransaction()` method marks the beginning of a new explicit transaction in the current connection. The `beginTransaction()`, `commit()` and `rollBack()` methods together provide management of transaction boundaries for an explicit transaction. This transaction contains SQL queries, which include the SQL statements INSERT, DELETE, or UPDATE, and a stored procedure that includes one of these SQL statements.

If you do *not* use `beginTransaction()` to specify the beginning of the explicit transaction, the database executes each SQL statement as a separate transaction.

Important: Only use `beginTransaction()` if the connection uses explicit transaction bracketing. If the connection uses implicit transaction bracketing, use of `beginTransaction()` results in a `CwDBTransactionException` exception.

Before beginning an explicit transaction, you must create a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class. Make sure that this connection uses explicit transaction bracketing.

Examples

The following example uses a transaction to execute a query for inserting rows into a table in the database associated with connections in the `CustDBConnPool`.

```
CwDBConnection connection = getDBConnection("CustDBConnPool", false);

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Commit the transaction
connection.commit();

// Release the connection
connection.release();
```

See also

`commit()`, `getDBConnection()`, `inTransaction()`, `rollBack()`

commit()

Commits the active transaction associated with the current connection.

Syntax

```
void commit()
```

Parameters

None.

Return values

None.

Exceptions

`CwDBConnectionException` – If a database error occurs.

Notes

The `commit()` method ends the active transaction by committing any changes made to the database associated with the current connection. The `beginTransaction()`, `commit()` and `rollBack()` methods together provide management of transaction boundaries for an explicit transaction. This transaction contains SQL queries, which include the SQL statements INSERT, DELETE, or UPDATE, and a stored procedure that includes one of these SQL statements.

Important: Only use `commit()` if the connection uses explicit transaction bracketing. If the connection uses implicit transaction bracketing, use of `commit()` results in a `CwDBTransactionException` exception. If you do not end an explicit transaction with `commit()` (or `rollback()`) before the connection is released, InterChange Server Express implicitly ends the transaction based on the success of the map. If the map is successful, ICS commits this database transaction. If the map is *not* successful, ICS implicitly rolls back the database transaction. Regardless of the success of the map, ICS logs a warning.

Before beginning an explicit transaction, you must create a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class. Make sure that this connection uses explicit transaction bracketing.

Examples

For an example of committing a transaction, see the example for `beginTransaction()`.

See also

`beginTransaction()`, `getDBConnection()`, `inTransaction()`, `rollBack()`

executePreparedSQL()

Executes a prepared SQL query by specifying its syntax and an optional parameter array.

Syntax

```
void executePreparedSQL(String query)
void executePreparedSQL(String query, Vector queryParameters)
```

Parameters

query — A string representation of the SQL query to execute in the database.

queryParameters — A `Vector` object of arguments to pass to parameters in the SQL query.

Return values

None.

Exceptions

`CwDBSQLException` – If a database error occurs.

Notes

The `executePreparedSQL()` method sends the specified *query* string as a prepared SQL statement to the database associated with the current connection. The first time it executes, this query is sent as a string to the database, which compiles the string into an executable form (called a prepared statement), executes the SQL statement, and returns this prepared statement to `executePreparedSQL()`. The `executePreparedSQL()` method saves this prepared statement in memory. Use `executePreparedSQL()` for SQL statements that you need to execute multiple times. The `executeSQL()` method does *not* save the prepared statement and is therefore useful for queries you need to execute only once.

Important: Before executing a query with `executePreparedSQL()`, you must obtain a connection to the desired database by generating a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class.

The SQL statements you can execute include the following (as long as you have the necessary database permissions):

- The SELECT statement to request data from one or more database tables

  Use the `hasMoreRows()` and `nextRow()` methods to access the retrieved data.
- SQL statements that modify data in the database
  - INSERT
  - DELETE
  - UPDATE

  If the connection uses explicit transaction bracketing, you must explicitly start each transaction with `beginTransaction()` and end it with either `commit()` or `rollback()`.
- The CALL statement to execute a prepared stored procedures with the limitation that this stored procedure *cannot* use any OUT parameters

  To execute stored procedures with OUT parameters, use the `executeStoredProcedure()` method.

See also

`beginTransaction()`, `commit()`, `executeSQL()`, `executeStoredProcedure()`, `getDBConnection()`, `hasMoreRows()`, `nextRow()`, `rollBack()`

---

executeSQL()

Executes a static SQL query by specifying its syntax and an optional parameter array.

Syntax

```
void executeSQL(String query)
void executeSQL(String query, Vector queryParameters)
```

Parameters

query — A string representation of the SQL query to execute in the database.

queryParameters — A `Vector` object of arguments to pass to parameters in the SQL query.

Return values

None.

Exceptions

`CwDBSQLException` – If a database error occurs.

Notes

The `executeSQL()` method sends the specified *query* string as a static SQL statement to the database associated with the current connection. This query is sent as a string to the database, which compiles the string into an executable form and executes the SQL statement, without saving this executable form. Use `executeSQL()` for SQL statements that you need to execute only once. The `executePreparedSQL()` method saves the executable form (called a prepared statement) and is therefore useful for queries you need to execute multiple times.

Important: Before executing a query with `executeSQL()`, you must obtain a connection to the desired database by generating a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class.

The SQL statements you can execute include the following (as long as you have the necessary database permissions):

- The SELECT statement to request data from one or more database tables

  Use the `hasMoreRows()` and `nextRow()` methods to access the retrieved data.
- SQL statements that modify data in the database
  - INSERT
  - DELETE
  - UPDATE

  If the connection uses explicit transaction bracketing, you must explicitly start each transaction with `beginTransaction()` and end it with either `commit()` or `rollback()`.
- The CALL statement to statically execute a stored procedures with the limitation that this stored procedure *cannot* use any OUT parameters

  To execute stored procedures with OUT parameters, use the `executeStoredProcedure()` method.

Examples

The following example executes a query for inserting rows into an accounting database whose connections reside in the `AccntConnPool` connection pool.

```
CwDBConnection connection = getDBConnection("AccntConnPool");

// Begin a transaction
connection.beginTransaction();

// Insert a row
connection.executeSQL("insert...");

// Commit the transaction
connection.commit();

// Release the database connection
connection.release();
```

For a more complete code sample that selects data from a relationship table, see

See also

`executePreparedSQL()`, `executeStoredProcedure()`, `getDBConnection()`, `hasMoreRows()`, `nextRow()`

executeStoredProcedure()

Executes an SQL stored procedure by specifying its name and parameter array.

Syntax

```
void executeStoredProcedure(String storedProcedure,
                  Vector storedProcParameters)
```

Parameters

storedProcedure
: The name of the SQL stored procedure to execute in the database.

storedProcParameters
: A `Vector` object of parameters to pass to the stored procedure. Each parameter is an instance of the `CwDBStoredProcedureParam` class. For more information on how to pass parameters through this array, see

Return values

None.

Exceptions

`CwDBSQLException` – If a database error occurs.

Notes

The `executeStoredProcedure()` method sends a call to the specified *storedProcedure* to the database associated with the current connection. This method sends the stored-procedure call as a prepared SQL statement; that is, the first time it executes, this stored-procedure call is sent as a string to the database, which compiles the string into an executable form (called a prepared statement), executes the SQL statement, and returns this prepared statement to `executeStoredProcedure()`. The `executeStoredProcedure()` method saves this prepared statement in memory.

Important: Before executing a stored procedure with `executeStoredProcedure()`, you must create a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class.

To handle any data that the stored procedure returns, use the `hasMoreRows()` and `nextRow()` methods.

You can also use the `executeSQL()` or `executePreparedSQL()` method to execute a stored procedure as long as this stored procedure does *not* contain OUT parameters. If the stored procedure uses OUT parameters, you *must* use `executeStoredProcedure()` to execute it. Unlike with `executeSQL()` or `executePreparedSQL()`, you do not have to pass in the full SQL statement to execute the stored procedure. With `executeStoredProcedure()`, you need to pass in only the name of the stored procedure and a `Vector` parameter array of `CwDBStoredProcedureParam` objects. The `executeStoredProcedure()` method can

determine the number of parameters from the *storedProcParameters* array and builds the calling statement for the stored procedure.

See also

executePreparedSQL(), executeSQL(), getDBConnection(), hasMoreRows(), nextRow()

getUpdateCount()

Returns the number of rows affected by the last write operation to the database.

Syntax

```
int getUpdateCount()
```

Parameters

None.

Return values

Returns an int representing the number of rows affected by the last write operation.

Exceptions

CwDBConnectionException – If a database error occurs.

Notes

The getUpdateCount() method indicates how many rows have been modified by the most recent update operation in the database associated with the current connection. This method is useful after you send an UPDATE or INSERT statement to the database and you want to determine the number of rows that the SQL statement has affected.

Important: Before using this method, you must create a CwDBConnection object with the getDBConnection() method from the BaseDLM class and send a query that updates the database with either the executeSQL() or executePreparedSQL() method from the CwDBConnection class.

See also

executePreparedSQL(), executeSQL(), getDBConnection()

hasMoreRows()

Determines whether the query result has more rows to process.

Syntax

```
boolean hasMoreRows()
```

Parameters

None.

Return values

Returns true if more rows exist.

Exceptions

`CwDBSQLException` – If a database error occurs.

Notes

The `hasMoreRows()` method determines whether the query result associated with the current connection has more rows to be processed. Use this method to retrieve results from a query that returns data. Such queries include a SELECT statement and a stored procedure. Only one query can be associated with the connection at a time. Therefore, if you execute another query before `hasMoreRows()` returns `false`, you lose the data from the initial query.

See also

`executePreparedSQL()`, `executeSQL()`, `nextRow()`

inTransaction()

Determines whether a transaction is in progress in the current connection.

Syntax

```
boolean inTransaction()
```

Parameters

None.

Return values

Returns `true` if a transaction is currently active in the current connection; returns `false` otherwise.

Exceptions

`CwDBConnectionException` – If a database error occurs.

Notes

The `inTransaction()` method returns a boolean value that indicates whether the current connection has an active transaction; that is, a transaction that has been started but not ended.

Important: Before beginning a transaction, you must create a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class.

See also

`beginTransaction()`, `commit()`, `getDBConnection()`, `rollBack()`

isActive()

Determines whether the current connection is active.

Syntax

```
boolean isActive()
```

Parameters

None.

Return values

Returns `true` if the current connection is active; returns `false` if this connection has been released.

Exceptions

None.

See also

`getDBConnection()`, `release()`

---

nextRow()

Retrieves the next row from the query result.

Syntax

```
Vector nextRow()
```

Parameters

None.

Return values

Returns the next row of the query result as a `Vector` object.

Exceptions

`CwDBSQLException` – If a database error occurs.

Notes

The `nextRow()` method returns one row of data from the query result associated with the current connection. Use this method to retrieve results from a query that returns data. Such queries include a SELECT statement and a stored procedure. Only one query can be associated with the connection at a time. Therefore, if you execute another query before `nextRow()` returns the last row of data, you lose the query result from the initial query.

See also

`hasMoreRows()`, `executePreparedSQL()`, `executeSQL()`, `executeStoredProcedure()`

---

release()

Releases use of the current connection, returning it to its connection pool.

Syntax

```
void release()
```

Parameters

None.

Return values

None.

Exceptions

`CwDBConnectionException`

Notes

The `release()` method explicitly releases use of the current connection by the map instance. Once released, the connection returns to its connection pool, where it is available for other components (maps or collaborations) that require a connection to the associated database. If you do not explicitly release a connection, the map instance implicitly releases it at the end of the current map run. Therefore, you *cannot* save a connection in a static variable and reuse it.

Attention: Do *not* use the `release()` method if a transaction is currently active. With implicit transaction bracketing, ICS does not end the database transaction until it determines the success or failure of the map. Therefore, use of this method on a connection that uses implicit transaction bracketing results in a `CwDBTransactionException` exception. If you do not handle this exception explicitly, it also results in an automatic rollback of the active transaction. You can use the `inTransaction()` method to determine whether a transaction is active.

See also

`getDBConnection()`, `inTransaction()`, `isActive()`

---

rollBack()

Rolls back the active transaction associated with the current connection.

Syntax

```
void rollBack()
```

Parameters

None.

Return values

None.

Exceptions

`CwDBConnectionException` – If a database error occurs.

Notes

The `rollback()` method ends the active transaction by rolling back any changes made to the database associated with the current connection. The `beginTransaction()`, `commit()` and `rollBack()` methods together provide management of transaction boundaries for an explicit transaction. This transaction contains SQL queries, which include the SQL statements INSERT, DELETE, or UPDATE, and a stored procedure that includes one of these SQL statements. If the roll back fails, `rollback()` throws the `CwDBTransactionException` exception and logs an error.

Important: Only use `rollback()` if the connection uses explicit transaction bracketing. If the connection uses implicit transaction bracketing, use of `rollback()` results in a `CwDBTransactionException` exception. If you do not end an explicit transaction with `rollback()` (or `commit()`) before the connection is released, InterChange Server Express implicitly ends the transaction based on the success of the map. If the map is successful, ICS commits this database transaction. If the map is *not* successful, ICS implicitly rolls back the database transaction. Regardless of the success of the map, ICS logs a warning.

Before beginning an explicit transaction, you must create a `CwDBConnection` object with the `getDBConnection()` method from the `BaseDLM` class. Make sure that this connection uses explicit transaction bracketing.

See also

`beginTransaction()`, `commit()`, `getDBConnection()`, `inTransaction()`

Chapter 13. CwDBStoredProcedureParam class

A `CwDBStoredProcedureParam` object describes a single parameter for a stored procedure. Table 96 summarizes the methods in the `CwDBStoredProcedureParam` class.

Table 96. CwDBStoredProcedureParam method summary

Method	Description	Page
`CwDBStoredProcedureParam()`	Constructs a new instance of `CwDBStoredProcedureParam` that holds argument information for the parameter of a stored procedure.	285
`getParamType()`	Retrieves the in/out type of the current stored-procedure parameter as an integer constant.	286
`getValue()`	Retrieves the value of the current stored-procedure parameter.	287

CwDBStoredProcedureParam()

Constructs a new instance of `CwDBStoredProcedureParam` that holds argument information for the parameter of a stored procedure.

Syntax

```
CwDBStoredProcedureParam(int paramType, String paramValue);

CwDBStoredProcedureParam(int paramType, int paramValue);
CwDBStoredProcedureParam(int paramType, Integer paramValue);
CwDBStoredProcedureParam(int paramType, Long paramValue);

CwDBStoredProcedureParam(int paramType, double paramValue);
CwDBStoredProcedureParam(int paramType, Double paramValue);
CwDBStoredProcedureParam(int paramType, float paramValue);
CwDBStoredProcedureParam(int paramType, Float paramValue);
CwDBStoredProcedureParam(int paramType, BigDecimal paramValue);

CwDBStoredProcedureParam(int paramType, boolean paramValue);
CwDBStoredProcedureParam(int paramType, Boolean paramValue);

CwDBStoredProcedureParam(int paramType, java.sql.Date paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Time paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Timestamp paramValue);

CwDBStoredProcedureParam(int paramType, java.sql.Blob paramValue);
CwDBStoredProcedureParam(int paramType, java.sql.Clob paramValue);

CwDBStoredProcedureParam(int paramType, byte[] paramValue);
CwDBStoredProcedureParam(int paramType, Array paramValue);
CwDBStoredProcedureParam(int paramType, Struct paramValue);
```

Parameters

paramType
The in/out parameter type of the associated stored-procedure parameter.

paramValue
The argument value to send to the stored procedure. This value is one of the following Java data types

© Copyright IBM Corp. 2003

Return values

Returns a new `CwDBStoredProcedureParam` object to hold the argument information for one argument in the declaration of the stored procedure.

Exceptions

None.

Notes

The `CwDBStoredProcedureParam()` constructor creates a `CwDBStoredProcedureParam` instance to describe one parameter for a stored procedure. Parameter information includes the following:

- The parameter's in/out type

  The constructor's first argument initializes this in/out parameter type. For a list of valid in/out parameter types, see Table 97.
- The parameter value

  The constructor's second argument initializes this parameter value. The `CwDBStoredProcedureParam` class provides one form of its constructor for each of the parameter-value data types it supports.

You provide a Java `Vector` of stored-procedure parameters to the `executeStoredProcedure()` method, which creates a stored-procedure call from a stored-procedure name and the parameter vector, and sends this call to the database associated with the current connection.

See also

`executeStoredProcedure()`

getParamType()

Retrieves the in/out type of the current stored-procedure parameter as an integer constant.

Syntax

```
int getParamType()
```

Parameters

None.

Return values

Returns the in/out type of the associated `CwDBStoredProcedureParam` parameter.

Exceptions

None.

Notes

The `getParamType()` method returns the in/out parameter type of the current stored-procedure parameter. The in/out parameter type indicates how the stored procedure uses the parameter. The `CwDBStoredProcedureParam` class represents each in/out type as a constant, as Table 97 shows.

Table 97. Parameter In/Out Types

Parameter in/out type	Description	In/Out type constant
IN parameter	An IN parameter is *input only*; that is, the stored procedure accepts its value as input but does *not* use the parameter to return a value.	`PARAM_IN`
OUT parameter	An OUT parameter is *output only*; that is, the stored procedure does *not* read its value as input but does use the parameter to return a value.	`PARAM_OUT`
INOUT parameter	An INOUT parameter is *input and output*; that is, the stored procedure accepts its value as input and also uses the parameter to return a value.	`PARAM_INOUT`

See also

`CwDBStoredProcedureParam()`, `getValue()`

getValue()

Retrieves the value of the current stored-procedure parameter.

Syntax

```
Object getValue()
```

Parameters

None.

Return values

Returns the value of the associated `CwDBStoredProcedureParam` parameter as a Java `Object`.

Exceptions

None.

Notes

The `getValue()` method returns the parameter value as a Java `Object` (such as `Integer`, `Double`, or `String`). If the value returned to an OUT parameter is the JDBC `NULL`, `getParamValue()` returns the `null` constant.

See also

`CwDBStoredProcedureParam()`, `getParamType()`

Chapter 14. DtpConnection class

The `DtpConnection` class is part of the Data Transformation Package (DTP). It provides methods for executing SQL queries on the relationship database. To instantiate this class, you must call `getRelConnection()` in the `BaseDLM` class. All maps are derived or subclassed from `BaseDLM` so they have access to `getRelConnection()`.

Important: The `DtpConnection` class and its methods are supported for backward compatibility *only*. These *deprecated methods* will not generate errors, but you should avoid using them and migrate existing code to the new methods. The deprecated methods might be removed in a future release. In new map development, use the `CwDBConnection` class and its methods to establish a database connection.

Table 98 summarizes the methods in the `DtpConnection` class.

Table 98. DtpConnection method summary

Method	Description	Page
`beginTran()`	Begins an SQL transaction for the relationship database.	289
`commit()`	Commits the current transaction in the relationship database.	290
`executeSQL()`	Executes a SQL query in the relationship database by specifying a CALL statement.	291
`execStoredProcedure()`	Executes an SQL stored procedure in the relationship database by specifying its name and parameter array.	292
`getUpdateCount()`	Returns the number of rows affected by the last write operation to the relationship database.	293
`hasMoreRows()`	Determines whether the query result has more rows to process.	293
`inTransaction()`	Determines whether a transaction is in progress in the relationship database.	294
`nextRow()`	Retrieves the next row in the query result vector.	294
`rollBack()`	Rolls back the current transaction in the relationship database.	295

beginTran()

Begins an SQL transaction for the relationship database.

Syntax

```
void beginTran()
```

Parameters

None.

Return values

None.

Exceptions

DtpConnectionException – If a database error occurs.

Notes

The beginTran(), commit() and rollBack() methods together provide transaction support for SQL queries.

Before beginning a transaction, you must create a DtpConnection object with the getRelConnection() method from the BaseDLM class.

Examples

The following example uses a transaction to execute a query for inserting rows into a table in the SapCust relationship.

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();
```

See also

commit(), getRelConnection(), inTransaction(), rollBack()

commit()

Commits the current transaction in the relationship database.

Syntax

```
void commit()
```

Parameters

None.

Return values

None.

Exceptions

DtpConnectionException – If a database error occurs.

Notes

The beginTran(), commit() and rollBack() methods together provide transaction support for SQL queries.

Before beginning a transaction, you must create a DtpConnection object with the getRelConnection() method from the BaseDLM class.

Examples

The following example uses a transaction to execute a query for inserting rows into a table in the SapCust relationship.

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();

// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();
```

See also

`beginTran()`, `getRelConnection()`, `inTransaction()`, `rollBack()`

---

executeSQL()

Executes a SQL query in the relationship database by specifying a CALL statement.

Syntax

```
void executeSQL(String query)
void executeSQL(String query, Vector queryParameters)
```

Parameters

query The SQL query to run in the relationship database.

queryParameters
A Vector object of arguments to pass to parameters in the SQL query.

Return values

None.

Exceptions

`DtpConnectionException` – If a database error occurs.

Notes

Before executing a query with `executeSQL()`, you must create a `DtpConnection` object with the `getRelConnection()` method from the `BaseDLM` class.

The SQL statements you can execute include INSERT, SELECT, DELETE, and UPDATE. You can also execute stored procedures with the limitation that this stored procedure *cannot* use any OUT parameters. To execute stored procedures with OUT parameters, use the `execStoredProcedure()` method.

Examples

The following example executes a query for inserting rows into a table in the `SapCust` relationship.

```
DtpConnection connection = getRelConnection("SapCust");

// begin a transaction
connection.beginTran();
```

```
// insert a row
connection.executeSQL("insert...");

// commit the transaction
connection.commit();

// release the database connection
releaseRelConnection(true);
```

See also

execStoredProcedure(), getRelConnection(), hasMoreRows(), nextRow()

execStoredProcedure()

Executes an SQL stored procedure in the relationship database by specifying its name and parameter array.

Syntax

```
void execStoredProcedure(String storedProcedure,
   Vector storedProcParameters)
```

Parameters

storedProcedure
: The name of the SQL stored procedure to run in the relationship database.

storedProcParameters
: A Vector object of parameters to pass to the stored procedure. Each parameter is an instance of the UserStoredProcedureParam class. For more information on how to pass parameters through this array, see

Return values

None.

Exceptions

DtpConnectionException – If a database error occurs.

Notes

Before executing a stored procedure with execStoredProcedure(), you must create a DtpConnection object with the getRelConnection() method from the BaseDLM class.

You can also use the executeSQL() method to execute a stored procedure as long as this stored procedure does not contain OUT parameters. If the stored procedure uses OUT parameters, you *must* use execStoredProcedure() to execute it. Unlike with executeSQL(), you do not have to pass in the full SQL statement to execute the stored procedure. With execStoredProcedure(), you need to pass in only the name of the stored procedure and a Vector parameter array of UserStoredProcedureParam objects. The execStoredProcedure() method can determine the number of parameters from the *storedProcParameters* array and builds the calling statement for the stored procedure.

See also

`executeSQL()`, `getRelConnection()`, `hasMoreRows()`, `nextRow()`

getUpdateCount()

Returns the number of rows affected by the last write operation to the relationship database.

Syntax

```
int getUpdateCount()
```

Parameters

None.

Return values

Returns an `int` representing the number of rows affected by the last write operation.

Exceptions

`DtpConnectionException` – If a database error occurs.

Notes

Before using this method, you must create a `DtpConnection` object with the `getRelConnection()` method from the `BaseDLM` class.

This method is useful after you send an UPDATE or INSERT statement on the relationship database and you want to determine the number of rows that the SQL statement has affected.

See also

`executeSQL()`, `getRelConnection()`

hasMoreRows()

Determines whether the query result has more rows to process.

Syntax

```
boolean hasMoreRows()
```

Parameters

None.

Return values

Returns `true` if more rows exist.

Exceptions

`DtpConnectionException` – If a database error occurs.

Notes

The `hasMoreRows()` method determines whether the query associated with the current relationship database has more rows to be processed. Use this method to retrieve results from a query that returns data. Such queries include a SELECT statement and a stored procedure. Only one query can be associated with the connection at a time. Therefore, if you execute another query before `hasMoreRows()` returns `false`, you lose the data from the initial query.

See also

`nextRow()`, `executeSQL()`, `getUpdateCount()`

inTransaction()

Determines whether a transaction is in progress in the relationship database.

Syntax

```
boolean inTransaction()
```

Parameters

None.

Return values

Returns `"True"` if a transaction is in progress.

Exceptions

`DtpConnectionException` – If a database error occurs.

Notes

Before beginning a transaction, you must create a `DtpConnection` object with the `getRelConnection()` method from the `BaseDLM` class.

See also

`beginTran()`, `commit()`, `getRelConnection()`, `rollBack()`

nextRow()

Retrieves the next row in the query result vector.

Syntax

```
Vector nextRow()
```

Parameters

None.

Return values

Returns the next row of the query result as a `Vector` object.

Exceptions

`DtpConnectionException` – If a database error occurs.

Notes

The `nextRow()` method returns one row of data from the query associated with the current relationship database. Use this method to retrieve results from a query that returns data. Such queries include a SELECT statement and a stored procedure. Only one query can be associated with the connection at a time. Therefore, if you execute another query before `nextRow()` returns the last row of data, you lose the data from the initial query.

See also

`hasMoreRows()`, `executeSQL()`, `getUpdateCount()`

---

rollBack()

Rolls back the current transaction in the relationship database.

Syntax

```
void rollBack()
```

Parameters

None.

Return values

None.

Exceptions

`DtpConnectionException` – If a database error occurs.

Notes

The `beginTran()`, `commit()` and `rollBack()` methods together provide transaction support for SQL queries.

Before beginning a transaction, you must create a `DtpConnection` object with the `getRelConnection()` method from the `BaseDLM` class.

See also

`beginTran()`, `commit()`, `getRelConnection()`, `inTransaction()`

Chapter 15. DtpDataConversion class

One of the most common tasks in business object mapping is the conversion of attribute values from one data type to another, a process called *data conversion*. The `DtpDataConversion` class provides a simple way to perform data conversions.

The data type classes in the `java.lang` package contain some conversion methods, but all possible conversions are not supported. The `DtpDataConversion` class consolidates many data conversion methods into one class and it supports the most common conversions that you perform in maps. The `getType()` and `isOKToConvert()` methods make it easy to determine whether specific conversions are possible.

All methods in this class are declared as static. Table 99 summarizes the methods of the `DtpDataConversion` class.

Table 99. DtpDataConversion method summary

Method	Description	Page
`getType()`	Determines the data type of a value.	297
`isOKToConvert()`	Determines whether it is possible to convert a value from one data type to another.	298
`toBoolean()`	Converts a Java object to a `Boolean` object.	300
`toDouble()`	Converts an object or primitive data type to a `Double` object.	301
`toFloat()`	Converts an object or primitive data type to a `Float` object.	301
`toInteger()`	Converts an object or primitive data type to an `Integer` object.	302
`toPrimitiveBoolean()`	Converts a String or Boolean object to the primitive `boolean` data type.	303
`toPrimitiveDouble()`	Converts an object or primitive data type to the primitive `double` data type.	303
`toPrimitiveFloat()`	Converts an object or primitive data type to the primitive `float` data type.	304
`toPrimitiveInt()`	Converts an object or primitive data type to the primitive `int` data type.	304
`toString()`	Converts an object or primitive data type to a String object.	305

getType()

Determines the data type of a value.

Syntax

```
int getType(Object objectData)
int getType(int integerData)
int getType(float floatData)
int getType(double doubleData)
int getType(boolean booleanData)
```

Parameters

objectData Any Java object.

© Copyright IBM Corp. 2003

integerData Any primitive `int` variable.

floatData Any primitive `float` variable.

doubleData Any primitive `double` variable.

booleanData Any primitive `boolean` variable.

Return values

Returns an integer representing the data type of the parameter you pass. You can interpret the return value by comparing it to one of these constants which are declared as static and final in the `DtpDataConversion` class:

INTEGER_TYPE The data is a primitive `int` value or `Integer` object.

STRING_TYPE The data is a `String` object.

FLOAT_TYPE The data is a primitive `float` value or `Float` object.

DOUBLE_TYPE The data is a primitive `double` value or `Double` object.

BOOL_TYPE The data is a primitive `boolean` value or `Boolean` object.

DATE_TYPE The data is a `Date` object.

LONGTEXT_TYPE The data is a `LongText` object.

UNKNOWN_TYPE The data is of an unknown type.

Exceptions

None.

Notes

You can use the return values from `getType()` in the `OKToConvert()` method to determine whether a conversion is possible between two given data types.

Examples

```
int conversionStatus = DtpDataConversion.isOKToConvert(
   DtpDataConversion.getType(srcObject),
   DtpDataConversion.getType(destObject));

switch(conversionStatus)
   {
   case DtpDataConversion.OKTOCONVERT:
      // go ahead and convert
      break;
   case DtpDataConversion.POTENTIALDATALOSS:
      // convert, then check value
      break;
   case DtpDataConversion.CANNOTCONVERT:
      // return an error
      break;
}
```

See also

`isOKToConvert()`

isOKToConvert()

Determines whether it is possible to convert a value from one data type to another.

Syntax

```
int isOKToConvert(int srcDatatype, int destDataType)
int isOKToConvert(String srcDataTypeStr, String destDataTypeStr)
```

Parameters

srcDataType
Integer returned by `getType()`, which represents the data type of the source value that you want to convert.

destDataType
Integer returned by `getType()`, which represents the data type to which you want to convert the source value.

srcDataTypeStr
String containing the data type name for the source value that you want to convert. Possible values are: `Boolean`, boolean, `Double`, `double`, `Float`, `float`, `Integer`, `int`, and `String`.

destDataTypeStr
String containing the data type name to which you want to convert the source value. Possible values are: `Boolean`, boolean, `Double`, `double`, `Float`, `float`, `Integer`, `int`, and `String`.

Return values

Returns an integer specifying whether it is possible to convert a value of the source data type to a value of the destination data type. You can interpret the return value by comparing it to one of these constants, which are declared as static and final in the `DtpDataConversion` class:

`OKTOCONVERT`
You can convert from the source to the destination data type.

`POTENTIALDATALOSS`
You can convert, but there is a potential for data loss if the source value contains unconvertable characters or must be truncated to fit the destination data type.

`CANNOTCONVERT`
The source data type cannot be converted to the destination data type.

Exceptions

None.

Notes

The `getType()` method returns an integer representing the data type of the value you pass as a parameter. You use the first form of `isOKToConvert()` together with `getType()` to determine whether a data conversion between two attributes is possible. In your `isOKToConvert()` method call, use `getType()` on both the source and destination attributes to generate the *srcDataType* and *destDataType* parameters.

The second form of the method accepts String values containing the data type names for the source and destination data. Use this form of the method if you know what the data types are, and you want to check whether you can perform a conversion.

Table 100 shows the possible conversions for each combination of source and destination data type. In the table:

- OK means you can convert the source type to the destination type with no data loss.
- DL means you can convert, but data loss might occur if the source contains unconvertable characters or must be truncated to fit the destination type.
- NO means you cannot convert the a value from source data type to the destination data type.

Table 100. Possible Conversions Between Data Types

	DESTINATION						
SOURCE	**int, Integer**	**String**	**float, Float**	**double, Double**	**boolean Boolean**	**Date**	**Longtext**
int Integer	OK	OK	OK	OK	NO	NO	OK
String	DL[1]	OK	DL[1]	DL[1]	DL[2]	DL	OK
float, Float	DL[3]	OK	OK	OK	NO	NO	OK
double, Double	DL[3]	OK	DL[3]	OK	NO	NO	OK
boolean, Boolean	NO	OK	NO	NO	OK	NO	OK
Date	NO	OK	NO	NO	NO	OK	OK
Longtext	DL[1]	DL[3]	DL[1]	DL[1]	DL[2]	DL	OK

[1]When converting a String or Longtext value to any numeric type, the String or Longtext value can contain only numbers and decimals. You must remove any other characters, such as currency symbols, from the String or Longtext value before converting. Otherwise, a `DtpIncompatibleFormatException` will be thrown.

[2]When converting a String or Longtext value to Boolean, the value of the String or Longtext should be `"true"` or `"false"`. Any string that is not `"true"` (case does not matter) will be considered false.

[3]Because the source data type supports greater precision than the destination data type, the value might be truncated.

Examples

```
if (DtpDataConversion.isOKToConvert(getType(mySource),
      getType(myDest))== DtpDataConversion.OKTOCONVERT)
   // map these attributes
else
   // skip these attributes
```

See also

`getType()`

toBoolean()

Converts a Java object to a `Boolean` object.

Syntax

```
Boolean toBoolean(Object objectData)
Boolean toBoolean(boolean booleanData)
```

Parameters

objectData A Java object that you want to convert to `Boolean`. The only object currently supported is `String`.

booleanData Any primitive `boolean` variable.

Return values

Returns a `Boolean` object.

Exceptions

`DtpIncompatibleFormatException` – If the source data type cannot be converted to Boolean.

Examples

```
Boolean MyBooleanObj = DtpDataConversion.toBoolean(MyStringObj);
```

See also

`getType()`, `isOKToConvert()`, `toPrimitiveBoolean()`

toDouble()

Converts an object or primitive data type to a `Double` object.

Syntax

```
Double toDouble(Object objectData)
Double toDouble(int integerData)
Double toDouble(float floatData)
Double toDouble(double doubleData)
```

Parameters

objectData	A Java object. The objects currently supported are: `Float`, `Integer`, and `String`.
integerData	Any primitive `int` variable.
floatData	Any primitive `float` variable.
doubleData	Any primitive `double` variable.

Return values

Returns a `Double` object.

Exceptions

`DtpIncompatibleFormatException` – If the source data type cannot be converted to Double.

Examples

```
Double myDoubleObj = DtpDataConversion.toDouble(myInteger);
```

See also

`getType()`, `isOKToConvert()`, `toPrimitiveDouble()`

toFloat()

Converts an object or primitive data type to a `Float` object.

Syntax

```
Float toFloat(Object objectData)
Float toFloat(int integerData)
Float toFloat(float floatData)
Float toFloat(double doubleData)
```

Parameters

objectData A Java object. The objects currently supported are: Double, Integer, and String.

integerData Any primitive int variable.

floatData Any primitive float variable.

doubleData Any primitive double variable.

Return values

Returns a Float object.

Exceptions

DtpIncompatibleFormatException – If the source data type cannot be converted to Float.

Examples

```
Float myFloatObj = DtpDataConversion.toFloat(myInteger);
```

See also

getType(), isOKToConvert(), toPrimitiveFloat()

---

toInteger()

Converts an object or primitive data type to an Integer object.

Syntax

```
Integer toInteger(Object objectData)
Integer toInteger(int integerData)
Integer toInteger(float floatData)
Integer toInteger(double doubleData)
```

Parameters

objectData A Java object. The objects currently supported are: Double, Float, and String.

integerData Any primitive int variable.

floatData Any primitive float variable.

doubleData Any primitive double variable.

Return values

Returns an Integer object.

Exceptions

DtpIncompatibleFormatException – If the source data type cannot be converted to Integer.

Examples

```
Integer myIntegerObj = DtpDataConversion.toInteger(myFloat);
```

See also

getType(), isOKToConvert(), toPrimitiveInt()

toPrimitiveBoolean()

Converts a String or Boolean object to the primitive boolean data type.

Syntax

```
boolean toPrimitiveBoolean(Object objectData)
```

Parameters

objectData A String or Boolean object that you want to convert to the primitive boolean data type.

Return values

Returns a primitive boolean value.

Exceptions

DtpIncompatibleFormatException – If the source data type cannot be converted to boolean.

Examples

```
boolean MyBoolean = DtpDataConversion.toPrimitiveBoolean(MyStringObj);
```

See also

getType(), isOKToConvert(), toBoolean()

toPrimitiveDouble()

Converts an object or primitive data type to the primitive double data type.

Syntax

```
double toPrimitiveDouble(Object objectData)
double toPrimitiveDouble(int integerData)
double toPrimitiveDouble(float floatData)
```

Parameters

objectData A Java object. The objects currently supported are: Double, Float, Integer, and String.

integerData Any primitive int variable.

floatData Any primitive float variable.

Return values

Returns a primitive `double` value.

Exceptions

`DtpIncompatibleFormatException` – If the source data type cannot be converted to double.

Examples

```
double myDouble = DtpDataConversion.toPrimitiveDouble(myObject);
```

See also

`getType()`, `isOKToConvert()`, `toDouble()`

toPrimitiveFloat()

Converts an object or primitive data type to the primitive `float` data type.

Syntax

```
float toPrimitiveFloat(Object objectData)
float toPrimitiveFloat(int integerData)
float toPrimitiveFloat(double doubleData)
```

Parameters

objectData	A Java object. The objects currently supported are: `Double`, `Float`, `Integer`, and `String`.
integerData	Any primitive `int` variable.
doubleData	Any primitive `double` variable.

Return values

Returns a primitive `float` value.

Exceptions

`DtpIncompatibleFormatException` – If the source data type cannot be converted to float.

Examples

```
float myFloat = DtpDataConversion.toPrimitiveFloat(myInteger);
```

See also

`getType()`, `isOKToConvert()`, `toFloat()`

toPrimitiveInt()

Converts an object or primitive data type to the primitive `int` data type.

Syntax

```
int toPrimitiveInteger(Object objectData)
int toPrimitiveInteger(float floatData)
int toPrimitiveInteger(double doubleData)
```

Parameters

objectData A Java object. The objects currently supported are: `Double`, `Float`, `Integer`, and `String`.

floatData Any primitive `float` variable.

doubleData Any primitive `double` variable.

Return values

Returns a primitive `int` value.

Exceptions

`DtpIncompatibleFormatException` – If the source data type cannot be converted to integer.

Examples

```
int myInt = DtpDataConversion.toPrimitiveInt(myObject);
```

See also

`getType()`, `isOKToConvert()`, `toInteger()`

toString()

Converts an object or primitive data type to a String object.

Syntax

```
String toString(Object objectData)
String toString(int integerData)
String toString(float floatData)
String toString(double doubleData)
```

Parameters

objectData A Java object. The objects currently supported are: `Double`, `Float`, and `Integer`.

integerData Any primitive `int` variable.

floatData Any primitive `float` variable.

doubleData Any primitive `double` variable.

Return values

Returns a `String` object.

Exceptions

`DtpIncompatibleFormatException` – If the source data type cannot be converted to String.

Examples

```
String myString = DtpDataConversion.toString(myObject);
```

See also

getType(), isOKToConvert()

Chapter 16. DtpDate class

The DtpDate class compares time and date values, sets their formats, and returns components of a time and date value.

The static (class) methods operate on the class name. The static methods take a set of business objects and return the earliest or latest dates or the business objects that contain the earliest or latest dates.

Instance methods operate on a date object. You pass a date value to the DtpDate constructor and you can then manipulate the resulting date object. Instance methods let you retrieve, format, and change the values associated with the date. You can also set the formats in which you want to handle dates.

The data conversion methods are useful when one application stores dates in one format and another application stores dates in another format. For example, SAP might send a date in the format 26/8/1999 15:23:20 but Clarify might need the date in the format August 26, 1999 15:23:20.

The values passed to the DtpDate class must follow these rules:

Day	A number from 1 to 30. If a separator between the month, year, and date is not present in the date-time string and the date is in a numeric format, single characters must be preceded by a zero (0), as in 01
Month	A number from 1 to 12, a name such as January or February, or an abbreviated (3 character) month name such as Jan or Feb. If a separator between the month, year, and date is not present in the date-time string and the date is in a numeric format, single characters must be preceded by a zero (0), as in 01.
Year	A 4-digit number.
Hour	A value in the range 01 to 23, representing 24-hour format. AM or PM designations are not allowed.
Minutes	A number in the range 01 to 59.
Seconds	A number in the range 01 to 59.

Table 101 summarizes the methods in the DtpDate class. Note that static and instance methods are separated in this table but are in alphabetical order in the chapter.

Table 101. DtpDate method summary

Method	Description	Page
Constructor		
DtpDate()	Parse the date according to the format specified.	309
Static methods		
getMaxDate()	From a list of business objects, return the latest date as a DtpDate object.	321
getMinDate()	From a list of business objects, return the earliest date as a DtpDate object.	323
getMaxDateBO()	From a list of business objects, return those that contain the latest date.	322
getMinDateBO()	From a list of business objects, return those that contain the earliest date.	325

© Copyright IBM Corp. 2003

Table 101. DtpDate method summary (continued)

Method	Description	Page
Instance methods		
addDays()	Add the specified number of days to this date.	310
addWeekdays()	Add the specified number of weekdays to this date.	311
addYears()	Add the specified number of years to this date.	312
after()	Check whether this date follows the date passed in as the input parameter.	313
before()	Check whether this date precedes the date passed in as the parameter.	314
calcDays()	Calculate the number of days between this date and another date.	314
calcWeekdays()	Calculate the number of weekdays between this date and another date.	315
get12MonthNames()	Return the current short-name representation of the twelve months for this date.	316
get12ShortMonthNames()	Return the current full-name representation of the twelve months for this date.	316
get7DayNames()	Return the current names for the seven days in the week for this date.	316
getCWDate()	Reformats this date into the IBM generic date format.	317
getDayOfMonth()	Return the day of the month for this date.	317
getDayOfWeek()	Return the day of the week for this date.	318
getHours()	Return the hours value for this date.	318
getIntDay()	Return the day of the week in this date as an integer.	318
getIntDayOfWeek()	Return the day of the week for this date.	319
getIntMilliSeconds()	Return the milliSeconds value from this date.	319
getIntMinutes()	Return the minutes value in this date as an integer.	319
getIntMonth()	Return the month in this date as an integer.	320
getIntSeconds()	Return the seconds in this date as an integer.	320
getIntYear()	Return the year in this date as an integer.	320
getMSSince1970()	Return the number of milliseconds between January 1, 1970 00:00:00 and this date.	321
getMinutes()	Return the minutes value from this date.	326
getMonth()	Return the full name representation of the month in this date.	326
getNumericMonth()	Return the month value from this date in numeric format.	326
getSeconds()	Return the seconds value from this date as a string.	327
getShortMonth()	Return the short name representation of the month name from this date.	327
getYear()	Return the year value in this date.	328
set12MonthNames()	Change the full-name representation for the twelve month names for this date.	328
set12MonthNamesToDefault()	Restore the full-name representation for the twelve month names to the default values for this date.	329
set12ShortMonthNames()	Change the short-name representation of the twelve month names for this date.	329
set12ShortMonthNamesToDefault()	Restore the short-name representation of the twelve month names to the default values for this date.	329

Table 101. DtpDate method summary (continued)

Method	Description	Page
set7DayNames()	Change the names of the seven days in the week for this date.	330
set7DayNamesToDefault()	Restore the names of the seven days in the week to the default values for this date.	330
toString()	Return the date in a specified format or the default format.	330

DtpDate()

Parse the date according to the format specified.

Syntax

```
public DtpDate()

public DtpDate(String dateTimeStr, String format)

public DtpDate(String dateTimeStr, String format, String[] monthNames,
   String[] shortMonthNames)

public DtpDate(long msSince1970, boolean isLocalTime)
```

Parameters

dateTimeStr
: The date-time in the form of a string.

format
: The date format. See Notes for details.

monthNames
: An array of strings representing the full 12 month names. If null, the default value is January, February, March, and so on.

shortMonthNames
: An array of strings representing the short month name. If this is null but *monthNames* is not null, this value is the first 3 letters of the full month names, such as Jan, Feb, Mar, Apr, and so on.

msSince1970
: The number of milliseconds since January 1, 1970 00:00:00.

isLocalTime
: Set this to `true` if the time is already a local time, or to `false` otherwise.

Return values

None

Exceptions

`DtpDateException` - When the constructor encounters parsing errors. This may occur if the date is not in the specified format.

Notes

The first form of the constructor does not take any parameters. It assigns the current date on the system to the new `DtpDate` object. It does not throw `DtpDateException`.

The second and the third forms of the constructor parse the date according to the specified date *format* and extract out the day, month, year, hour, minute, and second values. These can be retrieved and reformatted later with other DtpDate methods.

For example, a month can be retrieved in one of the following formats:
- The full-name representation (the default format): January, February, March, April, May, June, July, August, September, October, November, and December
- The numeric format: 1-12
- The short-name representation, which consists of the first three letters of each month name: Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec

The retrieved data does not depend of the context of the other data.

You can change the full-name and short-name representations of the month in the following ways:
- With the set12MonthNames() and set12ShortMonthNames()methods respectively
- By passing the representation as a parameter into the third form of the DtpDate() constructor

The fourth form of the constructor takes the number of milliseconds since January 1, 1970 00:00:00. Many applications represent the date in this manner.

Date format

In the date *format*, the date always precedes the time. The time is optional. If it is missing in a date-time string, the hours, minutes, and seconds have a default value of 00.

The date format uses the following case sensitive key letters:

D	day
M	month
Y	year
h	hours
m	minutes
s	seconds

These key letters may be separated by a separator such as "/" or "-".

Examples

The following examples show the DtpDate() constructor creating new date objects aDate, date2, and date3:

```
Dtpdate aDate = new DtpDate("5/21/1997 15:23:01", "M/D/Y h:m:s");
DtpDate date2 = new DtpDate("05211997 152301", "MDY hms");
DtpDate date3 = new DtpDate("Jan 10, 1999 10:00:00", "M D, Y h:m:s");
```

The following date format results in the DtpDateException being thrown:

```
h:m:s D/M/Y
```

addDays()

Add the specified number of days to this date.

Syntax

```
public DtpDate addDays(int numberOfDays)
```

Parameters

numberOfDays An integer number. If it is a negative number, the new date will be the date *numberOfDays* days before the current instance of `DtpDate`.

Return values

A new `DtpDate` object.

Exceptions

`DtpDateException`

Notes

The `addDays()` method adds the specified number of days to this date. You can use the `get()` methods to retrieve information about the resulting new date. The `DtpDate` object returned inherits all the properties of the current object of `DtpDate`, such as month names, date format, and so on.

The new date will be adjusted to be a valid date. For example, adding five days to January 29, 1999 00:00:00 results in February 03, 1999 00:00:00, and adding -30 days results in December 30, 1998 00:00:00.

Adding days does not affect the time of day.

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = toDay.addDays(1);
   System.out.println("Tomorrow is "
      + tomorrow.getDayOfMonth() + "/"
      + tomorrow.getNumericMonth() + "/"
      + tomorrow.getYear() + " "
      + tomorrow.getHours() + ":"
      + tomorrow.getMinutes() + ":"
      + tomorrow.getSeconds());
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

`addWeekdays()`, `addYears()`

---

addWeekdays()

Add the specified number of weekdays to this date.

Syntax

```
public DtpDate addWeekdays(int numberOfWeekdays)
```

Parameters

numberOfWeekdays
: An integer number. If it is a negative number, the new date will be the date that is *numberOfWeekdays* weekdays before the date represented by the current `DtpDate` object.

Return values

A new `DtpDate` object.

Exceptions

`DtpDateException`

Notes

The `addWeekdays()` method adds the specified number of weekdays to this date. You can then use the get methods to retrieve the information of the resulting new date. The `DtpDate` returned will inherit all the properties of the current instance of `DtpDate`, such as month names, date format, and so on.

Only Monday, Tuesday, Wednesday, Thursday, and Friday, or the equivalent values, are considered to be weekdays. Monday is considered to be the first day of the week.

Examples

```
try
   {
   DtpDate toDay = new DtpDate("8/2/1999 00:00:00", "M/D/Y h:m:s");
   DtpDate fiveWeekdaysLater = toDay.addWeekdays(5);
   // The new date should be 8/9/1999 00:00:00
   System.out.println("Next month is "
      + fiveWeekdaysLater.getDayOfMonth() + "/"
      + fiveWeekdaysLater.getNumericMonth() + "/"
      + fiveWeekdaysLater.getYear() + " "
      + fiveWeekdaysLater.getHours() + ":"
      + fiveWeekdaysLater.getMinutes() + ":"
      + fiveWeekdaysLater.getSeconds());
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

`addDays()`, `addYears()`

addYears()

Add the specified number of years to this date.

Syntax

```
public DtpDate addYears(int numberOfYears)
```

Parameters

numberOfYears An integer number. If it is a negative number, the new date will be the date that is *numberOfYears* years before the current `DtpDate` object.

Return values

A new `DtpDate` object.

Notes

The `addYears()` method adds the specified number of years to this date. You can then use the `get()` methods to retrieve the information of the resulting new date. The `DtpDate` returned inherits all the properties of the current instance of `DtpDate`, such as month names, date format, and so on.

Examples

```
DtpDate toDay = new DtpDate();
DtpDate lastYear= toDay.addYears(-1);
System.out.println("Next month is "
   + lastYear.getDayOfMonth() + "/"
   + lastYear.getNumericMonth() + "/"
   + lastYear.getYear() + " "
   + lastYear.getHours() + ":"
   + lastYear.getMinutes() + ":"
   + lastYear.getSeconds());
```

See also

`addDays()`, `addWeekdays()`

after()

Check whether this date follows the date passed in as the input parameter.

Syntax

```
public boolean after(DtpDate date)
```

Parameters

date The date to compare with this date.

Return values

Return `true` if this date follows the date passed in, and `false` if this date precedes the data passed in.

Exceptions

`DtpDateException`

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = yesterday.addDays(-1);
   // isAfter should be false.
   boolean isAfter = yesterday.after(today)
   }
```

```
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

before()

before()

Check whether this date precedes the date passed in as the parameter.

Syntax

```
public boolean before(DtpDate date)
```

Parameters

date The date to compare with this date.

Return values

Return `true` if this date precedes the date passed in, and `false` if this date follows the data passed in.

Exceptions

DtpDateException

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = yesterday.addDays(-1);
   // isBefore should be true.
   boolean isBefore = yesterday.before(today)
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

after()

calcDays()

Calculate the number of days between this date and another date.

Syntax

```
public int calcDays(DtpDate date)
```

Parameters

date The date to compare with this date.

Return values

An int representing the number of days. This is always a positive number.

Exceptions

`DtpDateException`

Notes

The `calcDays()` method calculates the difference in the number of days between this date and another date. The result is always a whole number of days.

The difference between 19990615 00:30:59 and 19990615 23:59:59 is 0 days, and the difference between 19990615 23:59:59 and 19990616 00:01:01 is 1 day.

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = toDay.addDays(1);
   int days = today.caldDays(tomorrow);
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

`calcWeekdays()`

---

calcWeekdays()

Calculate the number of weekdays between this date and another date.

Syntax

`public int calcWeekdays(DtpDate` *`date`*`)`

Parameters

date	The date to compare with this date.

Return values

An int representing the number of weekdays. This is always a positive number.

Exceptions

`DtpDateException`

Notes

The `calcWeekdays()` method calculates the number of weekdays between this date and another date. The difference between Friday and Saturday is 0, and between Friday and Monday is 1. Weekdays are assumed to be Monday through Friday or the equivalent values. A weekday is not the same as a business day, since a holiday can fall on a weekday.

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   DtpDate tomorrow = toDay.addDays(1);
   int days = today.caldWeekdays(tomorrow);
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

calcDays()

get12MonthNames()

Return the current full-name representation of the twelve months for this date.

Syntax

```
public String[ ] get12MonthNames()
```

Return values

An array of String objects containing the effective names of the twelve months.

Examples

```
DtpDate toDay = new DtpDate();
String[] toDay.get12MonthNames();
```

See also

set12MonthNames(), set12MonthNamesToDefault()

get12ShortMonthNames()

Return the current short-name representation of the twelve months for this date.

Syntax

```
public String[ ] get12ShortMonthNames()
```

Return values

An array of String objects containing the effective short names of the twelve months.

Examples

```
DtpDate toDay = new DtpDate();
String[] toDay.get12ShortMonthNames();
```

See also

set12ShortMonthNames(), set12ShortMonthNamesToDefault()

get7DayNames()

Return the current names for the seven days in the week for this date.

Syntax

```
public String[ ] get7DayNames()
```

Return values

An array of String objects containing the effective names for the seven days of the week.

Examples

```
DtpDate toDay = new DtpDate();
String[] toDay.get7DayNames();
```

See also

set7DayNames(), set7DayNamesToDefault()

getCWDate()

Reformats this date into the IBM generic date format.

Syntax

```
public String getCWDate()
```

Return values

A string representing the date in the IBM WebSphere InterChange Server Express generic business object format. The format is YMD hms. Examples of this format are:

- 19990615 150701
- 19990831 114122

Notes

The IBM generic date format takes the form:

YYYYMMDD HHMMSS

Examples

```
DtpDate toDay = new DtpDate();
String genericDate = toDay.getCWDate();
```

getDayOfMonth()

Return the day of the month for this date.

Syntax

```
public String getDayOfMonth()
```

Return values

The string representing the day of the month, such as 01, 20, 30, and so on.

Examples

```
DtpDate toDay = new DtpDate();
String dayOfMonth = toDay.getDayOfMonth();
```

See also

`getIntDay()`

getDayOfWeek()

Return the day of the week for this date.

Syntax

```
public String getDayOfWeek()
```

Return values

A string indicating day of the week, such as Monday, Tuesday, and so on.

Examples

```
DtpDate toDay = new DtpDate();
String dayOfWeek = toDay.getDayOfWeek();
```

See also

`getIntDayOfWeek()`

getHours()

Return the hours value for this date.

Syntax

```
public String getHours()
```

Return values

The string representing the hour value which will be between 00 and 23.

Examples

```
DtpDate toDay = new DtpDate();
String hours = toDay.getHours();
```

getIntDay()

Return the day of the month in this date as an integer.

Syntax

```
public int getIntDay()
```

Return values

An `int` value which is the day of the month.

Examples

```
DtpDate toDay = new DtpDate();
int day = toDay.getIntDay();
```

See also

`getDayOfMonth()`

getIntDayOfWeek()

Return the day of the week in this date as an integer.

Syntax

```
public int getIntDayOfWeek()
```

Return values

An int value which is the day of the week. The possible values are 0 (Monday), 1 (Tuesday), 2 (Wednesday), 3 (Thursday), 4 (Friday), 5 (Saturday), or 6 (Sunday).

Examples

```
DtpDate toDay = new DtpDate();
int dayOfWeek = toDay.getIntDayOfWeek();
```

See also

getDayOfWeek()

getIntMilliSeconds()

Return the milliSeconds value from the date.

Syntax

```
public int getIntMilliSeconds()
```

Return values

An int value which is the milliseconds The range is 0-999.

Examples

```
DtpDate toDay = new DtpDate();
int millisecs = toDay.getIntMilliSeconds();
```

getIntMinutes()

Return the minutes value in this date as an integer.

Syntax

```
public int getIntMinutes()
```

Return values

An int value which is the minutes. The range is 0-59.

Examples

```
DtpDate toDay = new DtpDate();
int mins = toDay.getIntMinutes();
```

See also

getMinutes()

getIntMonth()

Return the month in this date as an integer.

Syntax

```
public int getIntMonth()
```

Return values

An int value which is the month. The range is 1 (January) - 12 (December).

Examples

```
DtpDate toDay = new DtpDate();
int month = toDay.getIntMonth();
```

See also

getMonth(), getNumericMonth()

getIntSeconds()

Return the seconds in this date as an integer.

Syntax

```
public int getIntSeconds()
```

Return values

An int value which is the seconds. The range is 0-59.

Examples

```
DtpDate toDay = new DtpDate();
int secs = toDay.getIntSeconds();
```

See also

getSeconds(), getMSSince1970()

getIntYear()

Return the year in this date as an integer.

Syntax

```
public int getIntYear()
```

Return values

An int value which is the year.

Examples

```
DtpDate toDay = new DtpDate();
int year = toDay.getIntYear();
```

See also

getYear()

getMSSince1970()

Return the number of milliseconds between January 1, 1970 00:00:00 and this date.

Syntax

```
public long getMSSince1970()
```

Return values

An integer number. It may be negative if this date is before January 1, 1970 00:00:00.

Exceptions

DtpDateException

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   long ms = toDay.getMSSince1970();
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

See also

getSeconds()

getMaxDate()

From a list of business objects, return the latest date as a DtpDate object.

Syntax

```
public static DtpDate getMaxDate(BusObjArray boList, String attr,
      String dateFormat)
```

Parameters

boList A list of business objects.

attr The attribute of the business object to use when doing the comparison. The attribute must be of type Date.

dateFormat This is the date format. See DtpDate() for more details. If this is null, it is assumed that the date is the number of milliseconds since 1970.

Return values

A DtpDate object that contains the max date.

Exceptions

DtpIncompatibleBOTypeException - When the business objects in the list are not the same business object type.

DtpUnknownAttributeException - When the specified attribute is not a valid attribute in the business objects passed in.

DtpUnsupportedAttributeTypeException - When the type of the specified attribute is not one of the supported attribute types listed above.

All of these exceptions are subclasses of RunTimeEntityException.

Notes

The getMaxDate() method scans through the list of business objects looking for the business object with the latest date, and returns that date in the form of a DtpDate object.

Tip: This method is a static method.

In the date evaluation, Jan 1, 2004 000000 is later than Jan 1, 2002 000000, which is later than Jan 1, 1999 000000

The date information is assumed to be stored in the attribute name passed into the method. If an object has null date information, it is ignored. If all of the objects have null date information, null is returned.

Examples

```
try
   {
   DtpDate maxDate = DtpDate.getMaxDate(bos, "Start Date",
      "D/M/Y h:m:s");
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

See also

getMinDate(), getMaxDateBO()

getMaxDateBO()

From a list of business objects, return those that contain the latest date.

Syntax

```
public static BusObj[] getMaxDateBO(BusObj[] boList, String attr,
      String dateFormat)
public static BusObj[] getMaxDateBO(BusObjArray boList, String attr,
      String dateFormat)
```

Parameters

boList A list of business objects. It can be either an array of BusObj or an instance of BusObjArray. These business objects must be of the same business object type.

attr The attribute of the business object to compare with. The attribute must be of type Date.

dateFormat This is the date format. See `DtpDate()` for more details. If this is null, it is assumed that the date is the number of milliseconds since 1970.

Return values

An array of business objects that have the latest date.

Exceptions

All of these three exceptions are subclasses of `RunTimeEntityException`.

`DtpIncompatibleBOTypeException` - When the business objects in the list are not the same business object type.

`DtpUnknownAttributeException` - When the specified attribute is not a valid attribute in the business objects passed in.

`DtpUnsupportedAttributeTypeException` - When the type of the specified attribute is not one of the supported attribute types listed above.

`DtpDateException` - When the date format is invalid.

Notes

The `getMaxDateBO()` method scans through the list of business objects looking for the business object with the latest date and returns that business object. If multiple business objects have the same max date, all objects with that date are returned.

Tip: This method is a static method.

In the evaluation of which date is earliest, Jan 1, 2004 000000 is later than Jan 1, 2002 000000, which is later than Jan 1, 1999 000000.

The date information is assumed to be stored in the attribute name passed into the method. If an object has null date information, that object is ignored. If all of the objects have null date information, null is returned.

Examples

```
try
   {
   BusObj[] max = DtpDate.getMaxDateBO(bos, "Start Date",
      "D/M/Y h:m:s");
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

See also

`getMaxDate()`, `getMinDateBO()`

getMinDate()

From a list of business objects, return the earliest date as a `DtpDate` object.

Syntax

```
public static DtpDate getMinDate(BusObjArray boList, String attr,
      String dateFormat)
```

Parameters

boList A list of business objects.

attr The attribute of the business object to use when doing the comparison. The attribute must be of type `Date`.

dateFormat The date format. See `DtpDate()` for more details. If this is null, it is assumed that the date is the number of milliseconds since 1970.

Return values

A `DtpDate` object which contains the earliest date.

Exceptions

`DtpIncompatibleBOTypeException` - When the business objects in the list are not the same business object type.

`DtpUnknownAttributeException` - When the specified attribute is not a valid attribute in the business objects passed in.

`DtpUnsupportedAttributeTypeException` - When the type of the specified attribute is not one of the supported attribute types listed above.

All of these exceptions are subclasses of `RunTimeEntityException`.

Notes

The `getMinDate()` method scans through the list of business objects looking for the business object with the earliest date, and return that date in the form of a `DtpDate` object.

Tip: This method is a static method.

In the evaluation of dates, Jan 1, 1999 000000 is earlier than Jan 1, 2002 000000, which is earlier than Jan 1, 2004 000000.

The date information is assumed to be stored in the attribute name passed into the method. If an object has null date information, it is ignored. If all objects have null date information, null is returned.

Examples

```
try
   {
   DtpDate minDate = DtpDate.getMinDate(bos, "Start Date",
      "D/M/Y h:m:s");
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

See also

`getMaxDate()`, `getMinDateBO()`

getMinDateBO()

From a list of business objects, return those that contain the earliest date.

Syntax

```
public static BusObj[] getMinDateBO(BusObj[] boList, String attr,
      String dateFormat)
public static BusObj[] getMinDateBO(BusObjArray boList, String attr,
      String dateFormat)
```

Parameters

boList A list of business objects.

attr The attribute of the business object to use when doing the comparison. The attribute must be of type `Date`.

dateFormat The date format. See `DtpDate()` for more details. If this is null, it is assumed that the date is the number of milliseconds since 1970.

Return values

An array of business objects that have the date.

Exceptions

`DtpIncompatibleBOTypeException` - When the business objects in the list are not the same business object type.

`DtpUnknownAttributeException` - When the specified attribute is not a valid attribute in the business objects passed in.

`DtpUnsupportedAttributeTypeException` - When the type of the specified attribute is not one of the supported attribute types listed above.

`DtpDateException` - When the date format is invalid.

All of these exceptions are subclass of `RunTimeEntityException`.

Notes

The `getMinDateBO()` method scans through the list of business objects looking for the business object with the earliest date and returns that date in the form of a `DtpDate` object.

Tip: This method is a static method.

In the evaluation of the earliest date, Jan 1, 2004 000000 is later than Jan 1, 2002 000000 which is later than Jan 1, 1999 000000.

The date information is assumed to be stored in the attribute name passed into the method. If an object has null date information, it is ignored. If all of the objects have null date information, null is returned.

Examples

```
try
   {
   BusObj[] min = DtpDate.getMinDateBO(bos, "Start Date",
      "D/M/Y h:m:s");
```

```
   }
catch ( RunTimeEntityException err )
   {
   System.out.println(err.getMessage());
   }
```

See also

getMinDate(), getMaxDateBO()

getMinutes()

Return the minutes value from this date.

Syntax

```
public String getMinutes()
```

Return values

The string representing the minutes. The return value is between 00 and 59.

See also

getIntMinutes()

getMonth()

Return the full name representation of the month in this date.

Syntax

```
public String getMonth()
```

Return values

The name of the month, such as January, February, and so on.

See also

getIntMonth(), getNumericMonth(), getShortMonth()

getNumericMonth()

Return the month value from this date in numeric format.

Syntax

```
public String getNumericMonth()
```

Return values

The string in the numeric form for the month, such as 01, 02, and so on.

Examples

```
DtpDate toDay = new DtpDate();
System.out.println("Today is "
   + toDay.getDayOfMonth() + "/"
   + toDay.getNumericMonth() + "/"
```

```
   + toDay.getYear() + " "
   + toDay.getHours() + ":"
   + toDay.getMinutes() + ":"
   + toDay.getSeconds());
```

See also

getIntMonth(), getMonth()

getSeconds()

Return the seconds value from this date as a string.

Syntax

```
public String getSeconds()
```

Return values

The string representing the seconds. The return value is between 00 and 59.

Examples

```
DtpDate toDay = new DtpDate();
System.out.println("Today is "
   + toDay.getDayOfMonth() + "/"
   + toDay.getNumericMonth() + "/"
   + toDay.getYear() + " "
   + toDay.getHours() + ":"
   + toDay.getMinutes() + ":"
   + toDay.getSeconds());
```

See also

getIntSeconds()

getShortMonth()

Return the short name representation of the month name from this date.

Syntax

```
public String getShortMonth()
```

Return values

The name of the month in the short format, such as Jan, Feb, and so on.

Examples

```
DtpDate toDay = new DtpDate();
DtpDate lastYear= toDay.addYears(-1);
System.out.println("Next month is "
   + lastYear.getShortMonth() + " "
   + lastYear.getDayOfMonth() + ", "
   + lastYear.getYear() + " "
   + lastYear.getHours() + ":"
   + lastYear.getMinutes() + ":"
   + lastYear.getSeconds());
```

See also

getMonth(), set12ShortMonthNames(), set12ShortMonthNamesToDefault()

getYear()

Return the year value in this date.

Syntax

```
public String getYear()
```

Return values

The string representing the year. The year value includes the century. Examples are 1998 and 2004.

Examples

```
DtpDate toDay = new DtpDate();
DtpDate lastYear= toDay.addYears(-1);
System.out.println("Next month is "
   + lastYear.getDayOfMonth() + "/"
   + lastYear.getNumericMonth() + "/"
   + lastYear.getYear() + " "
   + lastYear.getHours() + ":"
   + lastYear.getMinutes() + ":"
   + lastYear.getSeconds());
```

See also

`getIntYear()`

set12MonthNames()

Change the full-name representation for the twelve month names for this date.

Syntax

```
public void set12MonthNames(String[] monthNames,
      boolean resetShortMonth)
```

Parameters

monthNames
An array of String containing the twelve month names. The first element is the first month of the year and the last element is the last month of the year.

resetShortMonthNames
By default, the short month names are the first three characters of the full month names. If this flag is set to `true`, the short month names will change based on the new full month names. If it is set to `false`, this method will not change the short month names.

Return values

None.

Exceptions

`DtpDateException` - When the month names passed in are not exactly 12 names.

See also

`get12MonthNames()`, `set12MonthNamesToDefault()`

set12MonthNamesToDefault()

Restore the full-name representation for the twelve month names to the default values for this date.

Syntax

```
public void set12MonthNamesToDefault()
```

Return values

None.

Notes

The default names are January, February, March, and so on.

See also

`get12MonthNames()`, `set12MonthNames()`

set12ShortMonthNames()

Change the short-name representation of the twelve month names for this date.

Syntax

```
public void set12ShortMonthNames(String[] shortMonths)
```

Parameters

shortMonths	A list of business objects.

Return values

None.

Exceptions

`DtpDateException` - When the month names passed in are not exactly 12 names.

See also

`get12ShortMonthNames()`, `set12ShortMonthNamesToDefault()`

set12ShortMonthNamesToDefault()

Restore the short-name representation of the twelve month names to the default values for this date.

Syntax

```
public void set12ShortMonthNamesToDefault()
```

Return values

None

Notes

The short month names are Jan, Feb, Mar, and so on.

See also

`get12ShortMonthNames()`, `set12ShortMonthNames()`

set7DayNames()

Change the names of the seven days in the week for this date.

Syntax

```
public void set7DayNames(String[] dayNames)
```

Parameters

dayNames An array of strings containing the seven days in a week. The first element should be the equivalent of Monday.

Return values

None.

Exceptions

`DtpDateException` - When exactly seven days are not specified.

See also

`get7DayNames()`, `set7DayNamesToDefault()`

set7DayNamesToDefault()

Restore the names of the seven days in the week to the default values for this date.

Syntax

```
public void set7DayNamesToDefault()
```

Return values

None.

Notes

The default names are Monday, Tuesday, Wednesday, and so on.

See also

`get7DayNames()`, `set7DayNames()`

toString()

Return the date in a specified format or the default format.

Syntax

```
public String toString()
public String toString(String format)
public String toString(String format boolean twelveHr)
```

Parameters

format	The date format. See `DtpDate()` for more details.
twelveHr	A boolean that, if set to true, specifies that the method returns 12-hour time instead of 24-hour time.

Return values

A string containing the date information, such as:

```
19990930 053029 PM
```

Regardless of the format of the month position, the output string is always a 2 character integer representation (that is, 01 for January, 12 for December, and so forth).

Exceptions

`DtpDateException` - When the date format is invalid.

Examples

```
try
   {
   DtpDate toDay = new DtpDate();
   String date = toDay.toString("Y/M/D h:m:s");
   }
catch ( DtpDateException date_e )
   {
   System.out.println(date_e.getMessage());
   }
```

Chapter 17. DtpMapService class

A submap is a map that you call from within another map. The `DtpMapService` class provides a method for running submaps. Table 102 summarizes the method in the `DtpMapService` class.

Table 102. DtpMapService method summary

Method	Description	Page
runMap()	Runs the map you specify.	333

runMap()

Runs the map you specify.

Syntax

```
BusObj[] runMap(String mapName, String mapType,
                BusObj[] srcBOs, cwExecCtx)
```

Parameters

mapName — The name of the map to run.

mapType — The type of the map to run. Use the following constant *only*, which is defined in the `DtpMapService` class: `CWMAPTYPE` – an IBM WebSphere InterChange Server Express map

srcBOs — An array of business objects that are the source business objects for *mapName*.

cwExecCtx — A variable that contains the execution context for the current map. This variable is defined in the code thatMap Designer Express generates for every map.

Return values

Returns an array of business objects that are the destination business objects of *mapName*.

Exceptions

`MapFailureException` – If an error occurs while attempting to run *mapName*.

`MapNotFoundException` – If *mapName* is not found in the repository.

`CxMissingIDException` – See `maintainSimpleIdentityRelationship()`.

Notes

Use the `runMap()` method to call a submap from within another map. For more information on calling submaps, see "Transforming with a submap" on page 41.

Examples

The following code calls a submap to map an application-specific `Address` business object to the generic `Address` business object:

© Copyright IBM Corp. 2003

```
// Create the BusObj Array
BusObj[] rSrcBOs = new BusObj[1];
rSrcBOs[0] = MyCustomerObj.MyAddressObj[0];

// Make the call to the map service
OutObjName = DtpMapService.runMap(MyAppAddressToGenAddress,
   DtpMapService.CWMAPTYPE,rSrcBOs,cwExecCtx);
```

See also

"Transforming with a submap" on page 41

Chapter 18. DtpSplitString class

The DtpSplitString class provides a way to split or parse a string into tokens and retrieve the results. This class is useful for manipulating formatted strings such as composite keys, dates, or telephone numbers.

DtpSplitString is similar to the StringTokenizer class in the java.util package. However, when working with IBM WebSphere InterChange Server Express maps, DtpSplitString provides these advantages over StringTokenizer:

- The tokens in a DtpSplitString object are indexed. This makes it easy to extract the specific tokens you are interested in. For example, if you parse a telephone number (such as 650-555-1111) into three tokens using the dash (-) as a delimiter, you can extract the area code by referencing element 0 and build the rest of the telephone number by concatenating element 1 and element 2.
- A DtpSplitString object allows bidirectional scrolling of the tokens. As you navigate the elements using nextElement() and prevElement() all the elements remain available.

Table 103 summarizes the methods in the DtpSplitString class.

Table 103. DtpSplitString method summary

Method	Description	Page
DtpSplitString()	Constructs a new instance of DtpSplitString and parses a string into tokens.	335
elementAt()	Returns an element in the DtpSplitString object at the position you specify.	336
firstElement()	Returns the element in the DtpSplitString object at position zero.	336
getElementCount()	Returns an integer containing the total number of elements.	337
getEnumeration()	Returns an Enumeration of String objects where each String is one of the parsed tokens.	338
lastElement()	Returns the last element in the DtpSplitString object.	338
nextElement()	Returns the next element in the DtpSplitString object.	338
prevElement()	Returns the previous element in the DtpSplitString object.	339
reset()	Resets the current position number in the DtpSplitString object to zero.	340

DtpSplitString()

Constructs a new instance of DtpSplitString and parses a string into tokens.

Syntax

```
DtpSplitString(String str, String delimiters)
```

Parameters

str The string to parse.

delimiters — A String containing the delimiters used in *str*. There can be more than one delimiter, but each delimiter can be only one character in length.

Notes

DtpSplitString() parses *str* into tokens, called elements, based on the specified delimiters. After calling DtpSplitString(), you can call any of the DtpSplitString class methods to select and retrieve specific elements.

Examples

```
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");
```

elementAt()

Returns an element in the DtpSplitString object at the position you specify.

Syntax

```
String elementAt(int nth)
```

Parameters

nth — The position of the element to extract from the DtpSplitString object. The position of the first element is zero.

Return values

Returns a String containing the element at the *nth* position.

Exceptions

DtpNoElementAtPositionException – If you specify an invalid position for *nth*.

Notes

Elements are numbered from first to last beginning with zero. For example, if the delimiters are commas and spaces, then the element at position two in the string, "This,is a test" is "a".

The elementAt() method returns the element at the specified position but does not change the current element position.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

//This call returns "a"
public String MyString.elementAt(2);
```

See also

getElementCount()

firstElement()

Returns the element in the DtpSplitString object at position zero.

Syntax

```
String firstElement()
```

Return values

Returns a String containing the element at position zero.

Exceptions

`DtpNoElementAtPositionException` – If there are no elements.

Notes

Elements in the `DtpSplitString` object are numbered from first to last beginning with zero. Therefore, the first element is at position zero.

The `firstElement()` method returns the element at position zero but does *not* change the current element position.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns the first element containing "This"
String anElement = MyString.firstElement();
```

See also

`lastElement()`

getElementCount()

Returns the total number of elements in the `DtpSplitString` object.

Syntax

```
int getElementCount()
```

Return values

Returns an integer containing the total number of elements.

Notes

Elements are numbered from first to last beginning with zero. If `getElementCount()` returns 6, the highest-numbered element is 5.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns the integer 4
String numElements = MyString.getElementCount();
```

See also

`firstElement()`, `lastElement()`

getEnumeration()

Returns an Enumeration of String objects where each String is one of the parsed tokens.

Syntax

```
Enumeration getEnumeration()
```

Return values

Returns an Enumeration object.

Notes

The `getEnumeration()` method provides another way to process the parsed tokens in a `DtpSplitString` object. For more information on working with Enumeration objects, see the `Java.Util` package.

lastElement()

Returns the last element in the `DtpSplitString` object.

Syntax

```
String lastElement()
```

Return values

Returns a String containing the last element.

Exceptions

`DtpNoElementAtPositionException` – If there are no elements.

Notes

Elements are numbered from first to last beginning with zero. The last element is the highest-numbered element. The position number of the last element is equivalent to `getElementCount()-1`.

The `lastElement()` method returns the last element but does not change the current element position.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns the last element, containing "test"
String anElement = MyString.lastElement();
```

See also

`firstElement()`, `getElementCount()`

nextElement()

Returns the next element in the `DtpSplitString` object.

Syntax

```
String nextElement()
```

Return values

Returns a String containing the next element.

Exceptions

`DtpNoElementAtPositionException` – If there is no next element.

Notes

The first time you call `nextElement()`, it returns the element at position zero. In subsequent method calls, `nextElement()` returns the element at position one, two, three, and so on. You can use `nextElement()`, along with `prevElement()`, to navigate the elements (tokens) in a `DtpSplitString` object.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()

// This call returns element 1 containing "is"
String secondElement = MyString.nextElement()
```

See also

`prevElement()`, `reset()`

---

prevElement()

Returns the previous element in the `DtpSplitString` object.

Syntax

```
String prevElement()
```

Return values

Returns a String containing the previous element.

Exceptions

`DtpNoElementAtPositionException` – If there is no previous element.

Notes

You can use `prevElement()`, along with `nextElement()`, to navigate the elements (tokens) in a `DtpSplitString` object. The first time you call `nextElement()`, the element position is zero. Subsequent calls to `nextElement()`increment the position by one. The `prevElement()` method returns the previous element and decrements the element position by one.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()

// This call returns element 1 containing "is"
String secondElement = MyString.nextElement()

// This call returns element 0 containing "This"
String anotherElement = MyString.prevElement()
```

See also

`nextElement()`

---

reset()

Resets the current position number in the `DtpSplitString` object to zero.

Syntax

```
void reset()
```

Return values

None.

Notes

The default element position is zero. Each time you call `nextElement()`, the element position increments by one. The `prevElement()` method returns the previous element and decrements the element position by one. You can use `reset()` to reset the current position back to zero.

Examples

```
// Create a DtpSplitString object
DtpSplitString MyString = new DtpSplitString("This,is a test",", ");

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()

// This call returns element 1 containing "is"
String secondElement = MyString.nextElement()

// Reset the position to zero
MyString.reset()

// This call returns element 0 containing "This"
String firstElement = MyString.nextElement()
```

See also

`nextElement()`, `prevElement()`

Chapter 19. DtpUtils class

The DtpUtils class performs several general-purpose operations.

Table 104 summarizes the methods of the DtpUtils class.

Table 104. DtpUtils method summary

Method	Description	Page
padLeft()	Pads the string with the specified character.	341
padRight()	Pads the string with the specified character.	341
stringReplace()	Replaces all occurrences of a pattern within a string with another pattern.	341
truncate()	Truncates this number.	343

padLeft()

Pads the string with the specified character.

Syntax

```
public static String padLeft(String src, char padWith, int totalLen)
```

Parameters

src The string to be padded.

padWith The character used in padding.

totalLen The new size of the string, a positive number. If the value is 0, smaller than the size of the original string, or a negative number, the original string is returned.

Return values

A new padded string.

Notes

Pads the string with a specified character.

Examples

The following call returns 0000012345:

```
padLeft("12345", '0', 10);
```

The following call returns 123456:

```
padLeft("123456", '0', 5);
```

padRight()

Pads the string with the specified character.

Syntax

```
public static String padLeft(String src, char padWith, int totalLen)
```

© Copyright IBM Corp. 2003

Parameters

src	The string to be padded.
padWith	The character used in padding.
totalLen	The new size of the string, a positive number. If the value is 0, smaller than the size of the original string, or a negative number, the original string is returned.

Return values

A new padded string.

Notes

Pads the string with a specified character.

Examples

The following call returns 1234500000:

```
padRight("12345", '0', 10);
```

The following call returns 123456:

```
padRight("123456", '0', 5);
```

stringReplace()

Replaces all occurrences of a pattern within a string with another pattern.

Syntax

```
public static String stringReplace(String src, String oldpattern,
     String newPattern)
```

Parameters

src	The string to change.
oldPattern	The character used in padding.
newPattern	The string pattern to use in replacement.

Return values

A new string with the new pattern.

Notes

The method replaces all occurrences of the value specified by *oldPattern* with the value specified by *newPattern*. For single character replacement, use the `replace()` in the Java `String` class. If *oldPattern* is not found, the original, unmodified string is returned.

Examples

The following results in `youoyou and dad`.

```
stringReplace("momomom and dad", "mom", "you");
```

truncate()

Truncates this number.

Syntax

```
public static double truncate(Object aNumber, int precision)
   throws DtpIncompatibleFormatException

public static double truncate(float aNumber, int precision)
public static double truncate(double aNumber, int precision)

public static int truncate(Object aNumber)
   throws DtpIncompatibleFormatException

public static int truncate(float aNumber)
public static int truncate(double aNumber)
```

Parameters

aNumber A number. The valid types are `String`, `float`, and `double`.

precision The number of digits to the right of the decimal to be removed.

Return values

A `double` or `int` number.

Notes

This method removes digits from this number, starting from the right.

The first three forms of the methods truncate the number by removing the digits to the right of the decimal place, starting from the right. If the input number is an integer, it will not get truncated. The number of type `Object` must be either `String`, `Double` or `Float`.

The last three forms of the methods truncate the number by removing all digits to the right of the decimal and return the int value.

Examples

The following returns 123.45:

```
truncate("123.4567", 2);
```

The following returns 123:

```
truncate(123.456, 4)
```

Chapter 20. IdentityRelationship class

The methods documented in this chapter operate on objects of the `IdentityRelationship` class. These objects represent instances of identity relationships. The `IdentityRelationship` class provides additional functionality needed when accessing the repository database. It combines a set of existing APIs into methods that provide ease of use for the map developer.

The source code for the methods in the `IdentityRelationship` class is provided and can be used as is in the IBM WebSphere InterChange Server Express environment, or can be customized to fit other environments.

Table 105 lists the methods of the `IdentityRelationship` class.

Table 105. IdentityRelationship method summary

Method	Description	Page
`addMyChildren()`	Adds the specified child instances to a parent/child relationship for an identity relationship.	345
`deleteMyChildren()`	Removes the specified child instances to a parent/child relationship for an identity relationship belonging to the specified parent.	347
`foreignKeyLookup()`	Performs a lookup in a foreign relationship table based on the foreign key of the source business object, failing to find a relationship instance if the foreign key does not exist in the foreign relationship table.	348
`foreignKeyXref()`	Performs a lookup in the relationship table in the relationship database based on the foreign key of the source business object, adding a new relationship instance in the foreign relationship table if the foreign key does not exist.	350
`maintainChildVerb()`	Sets the child business object verb based on the map execution context and the verb of the parent business object.	352
`maintainCompositeRelationship()`	Maintains a composite identity relationship from within the parent map.	354
`maintainSimpleIdentityRelationship()`	Maintains a simple identity relationship from within either a parent or child map.	356
`updateMyChildren()`	Adds and deletes child instances in a specified parent/child relationship of an identity relationship as necessary.	358

Note: All methods in the `IdentityRelationship` class are declared as static. You can call any of the methods in this class from an existing relationship instance or by referencing the `IdentityRelationship` class: `IdentityRelationship.`*`method`*, where *`method`* is the name of a method in Table 105.

addMyChildren()

Adds the specified child instances to a parent/child relationship for an identity relationship.

© Copyright IBM Corp. 2003

Syntax

```
public static void addMyChildren(String parentChildRelDefName,
      String parentParticpntDefName, BusObj parentBusObj,
      String childParticpntDefName, Object childBusObjList,
      CxExecutionContext map_ctx)
```

Parameters

parentChildRelDefName
: The name of the parent/child relationship definition.

parentParticpntDefName
: The name of the participant definition that represents the parent business object in the parent/child relationship.

parentBusObj
: The variable that contains the parent business object.

childParticpntDefName
: The name of the participant definition that represents the child business object in the parent/child relationship.

childBusObjList
: The variable that contains child business object or objects to be added to the relationship. This parameter can be either a single generic business object (`BusObj`) or an array of generic business objects (`BusObjArray`).

map_ctx
: The map execution context. To pass the map execution context, use the `cwExecCtx` variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

`RelationshipRuntimeException`

Notes

The `addMyChildren()` method adds the child instances in *childBusObjList* to the relationship tables of the *parentChildRelDefName* relationship definition. This method is useful in a custom relationship involving a parent business object with a unique key. When a parent business object has the addition of new child objects, use `addMyChildren()` to compare the after-image (in *parentBusObj*) with the before-image (information in the relationship tables) to determine which child objects in the after-image are new. For each new child object, `addMyChildren()` adds a child instance to the relationship tables for the parent and child participants (*parentParticpntDefName* and *childParticpntDefName*, respectively). If the parent business object does not exist in the relationship table, `addMyChildren()` inserts a relationship instance for this parent object.

The `addMyChildren()` method requires that a parent/child relationship be defined with Relationship Designer Express. For information on how to create this kind of relationship, see “Creating the parent/child relationship definition” on page 208..

See also

`deleteMyChildren()`, `updateMyChildren()`

"Managing child instances" on page 207,.

deleteMyChildren()

Removes the specified child instances to a parent/child relationship for an identity relationship belonging to the specified parent.

Syntax

```
void deleteMyChildren(String parentChildRelDefName,
      String parentParticpntDefName, BusObj parentBusObj,
      String childParticpntDefName, Object childBusObjList,
      CxExecutionContext map_ctx)

void deleteMyChildren(String parentChildRefDefName,
      String parentParticpntDefName, BusObj parentBusObj,
      String childParticpntDefName, CxExecutionContext map_ctx)
```

Parameters

parentChildRelDefName
: The name of the parent/child relationship definition.

parentParticpntDefName
: The name of the participant definition that represents the parent business object in the parent/child relationship.

parentBusObj
: The variable that contains the parent business object.

childParticpntDefName
: The name of the participant definition that represents the child business object in the parent/child relationship.

childBusObjList
: The variable that contains child business object or objects to be deleted from the relationship. This parameter can be either a single generic business object (`BusObj`) or an array of generic business objects (`BusObjArray`).

map_ctx
: The map execution context. To pass the map execution context, use the `cwExecCtx` variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

`RelationshipRuntimeException`

Notes

The `deleteMyChildren()` method deletes child instances from a parent/child *parentChildRelDefName* relationship definition. It supports the following forms:

- The first form of the method removes from the relationship tables for the parent and child participants those child instances that correspond to each of the child

business objects in *childBusObjList*. It locates a child instance to delete by matching the child object's value and name, as well as the parent object's value and name.

- The second form of the method removes from relationship tables for the parent and child participants all child instances for the *parentBusObj* parent object. It locates the child instance to delete by matching the parent object's value and name.

This method is useful in a custom relationship involving a parent business object with a unique key. When a parent business object has removed child objects, use `deleteMyChildren()` to compare the after-image (in *parentBusObj*) with the before-image (information in the relationship tables) to determine which child objects in the after-image have been removed. For each child object, `deleteMyChildren()` removes the corresponding child instance from the relationship tables for the parent and child participants (*parentParticpntDefName* and *childParticpntDefName*, respectively).

The `deleteMyChildren()` method requires that a parent/child relationship be defined with Relationship Designer Express. For information on how to create this kind of relationship, see "Creating the parent/child relationship definition" on page 208..

See also

`addMyChildren()`, `updateMyChildren()`

"Managing child instances" on page 207

foreignKeyLookup()

Performs a lookup in a foreign relationship table based on the foreign key of the source business object, failing to find a relationship instance if the foreign key does not exist in the foreign relationship table.

Syntax

```
public static void foreignKeyLookup(String relDefName,
      String appParticpntDefName, BusObj
 appSpecificBusObj,       String appForeignAttr,
       BusObj genericBusObj,
String genForeignAttr, CwExecutionContext map_ctx)
```

Parameters

relDefName — The name of the simple identity relationship that manages the foreign business object.

appParticpntDefName — The name of the participant definition that represents the application-specific business object in the simple identity relationship. The type of this participant is the foreign application-specific business object.

appSpecificBusObj — The variable that contains the application-specific business object, which contains the reference to the foreign business object.

appForeignAttr
: The name of the attribute in the application-specific business object that contains a key value for the foreign business object.

genericBusObj
: The variable that contains the generic business object to or from which the *appSpecificObject* is being mapped.

genForeignAttr
: The name of the attribute name in the generic business object that contains the generic reference to a foreign business object.

map_ctx
: The map execution context. To pass the map execution context, use the `cwExecCtx` variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

`RelationshipRuntimeException`

Notes

The `foreignKeyLookup()` method performs a foreign key lookup on the relationship table for the *AppParticpntDefName* participant; that is, it checks the foreign relationship table for a relationship instance that matches the value in the foreign key of the *appSpecificBusObj* business object. If this lookup fails, the `foreignKeyLookup()` method just sets the foreign key in the destination business object to null; it does *not* insert a row in the foreign relationship table (as the `foreignKeyXref()` method does). This method can be used in both inbound and outbound maps.

Examples

On the `Clarify_PartRequest` to `Requisition` object, the `VendorId` field is a foreign key lookup. This is because Purchasing does not call Vendor Wrapper. We do not use the `foreignKeyXref()` method here because we do not want to insert a row if the lookup fails.

```
if (ObjCustomerRole.isNull("RoleId"))
   {
   logError(5003, "OrderAssociatedCustomers.RoleId");
   // throw new MapFailureException("OrderAssociatedCustomers.RoleId
   // is null");
   }

try
   {
   IdentityRelationship.foreignKeyLookup("Customer", "SAPCust",
         ObjSAP_OrderPartners, "PartnerId", ObjCustomerRole,
         "RoleId", cwExecCtx);
   }

catch (RelationshipRuntimeException re)
   {
   logWarning(re.getMessage());
   }
if (ObjSAP_OrderPartners.get("PartnerId") == null)
   {
   logError(5007, "SAP_OrderPartners.PartnerId",
```

```
    "OrderAssociatedCustomers.RoleId", "Customer", "SAPCust",
    strInitiator);
 throw new MapFailureException("ForeignKeyLookup failed");
 }
```

See also

foreignKeyXref()

"Performing foreign key lookups" on page 216

foreignKeyXref()

Performs a lookup in the relationship table in the relationship database based on the foreign key of the source business object, adding a new relationship instance in the foreign relationship table if the foreign key does not exist.

Syntax

```
public static void foreignKeyXref(String relDefName,
      String appParticpntDefName, String genParticpntDefName,
      BusObj appSpecificBusObj, String appForeignAttr,
      BusObj genericBusObj, String genForeignAttr,
      CxExecutionContext map_ctx)
```

Parameters

relDefName
: The name of the simple identity relationship name that manages the foreign business object.

appParticpntDefName
: The name of the participant definition for the application-specific business object in the simple identity relationship. The type of this participant is the foreign application-specific business object.

genParticpntDefName
: The name of the participant definition for the generic business object in the simple identity relationship. The type of this participant is the foreign generic business object.

appSpecificBusbj
: The application-specific business object that contains the reference to the foreign object.

appForeignAttr
: The name of the attribute in the application-specific business object that contains a key value for the foreign business object.

genericObject
: The generic business object to or from which the *appSpecificObject* is being mapped.

genForeignAttr
: The name of the attribute name in the generic business object that contains the generic reference to a foreign business object.

map_ctx
: The map execution context. To pass the map execution context, use the cwExecCtx variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

RelationshipRuntimeException

Notes

The foreignKeyXref() method performs a foreign key lookup on the relationship table for the *AppParticpntDefName* participant; that is, it checks the foreign relationship table for a relationship instance that matches the value in the foreign key of the *appSpecificBusObj* business object. If this lookup fails, the foreignKeyXref() method adds a new relationship instance for the application-specific key to the foreign relationship table; it does *not* just set the foreign key in the destination business object to null (as the foreignKeyLookup() method does). This method can be used in both inbound and outbound maps.

The foreignKeyXref() method performs the following validations on arguments that are passed in:

- Validate the name of the *relDefName* relationship definition.
- Validate the name of the *particpntDefName* participant definition for the application-specific business object.
- Make sure that the *relDefName* relationship is an identity relationship. In addition, the participant definition in *relDefName* that represents the generic business object must be defined as IBM WebSphere InterChange Server Express-managed. For more information on how to specify these settings, see "Defining identity relationships" on page 174..

If any of these validations fails, foreignKeyXref() throws the RelationshipRuntimeException exception.

Once the arguments are validated, the action that foreignKeyXref() takes depends on the following information:

- The calling context—in the map execution context, passed in as part of the *map_ctx* argument (cwExecCtx)
- The verb—in the source business object
  - Application-specific business object (*appSpecificBusObj*) for calling contexts EVENT_DELIVERY (or ACCESS_REQUEST) and SERVICE_CALL_RESPONSE
  - Generic business object (*genericBusObj*) for calling contexts SERVICE_CALL_REQUEST and ACCESS_RESPONSE

The foreignKeyXref() method handles all of the basic adding of relationship instances in the foreign relationship table for the appropriate combination of calling context and verb. For more information on the actions that foreignKeyXref() takes, see "Using the Foreign Key Cross-Reference function block" on page 217.. Table 89 and Table 90 provide the actions for each of the calling contexts.

Examples

On the Clarify_SFAQuote to Order map, the CustomerId field is a foreign key lookup. This is because Sales Order Processing Collab calls Customer Wrapper.

```
if (ObjSAP_OrderLineItem.get("SAP_OrderLineObjectIdentifier[0]")
      != null)
   {
   if (ObjSAP_OrderLineItem.getString(
"SAP_OrderLineObjectIdentifier[0].ObjectQualifier").equals("002"))
      {
      BusObj temp = ObjSAP_OrderLineItem.getBusObj(
```

```
         "SAP_OrderLineObjectIdentifier[0]");
      if (temp.isNull("ItemId"))
         {
         logWarning(5003,
    "SAP_OrderLineItem.SAP_OrderLineObjectIdentifier[1].ItemId");
         }
      else
         {
         try
            {
            IdentityRelationship.foreignKeyXref(
               "Item",
               "SAPMbasc",
               "CWItba",
               temp,
               "ItemId",
               ObjOrderLineItem,
               "ItemId",
               cwExecCtx);
            }
         catch (RelationshipRuntimeException re)
            {
            logWarning(re.getMessage());
            }

         if (ObjOrderLineItem.get("ItemId") == null )
            {
            logError(5009, "OrderLineItem.ItemId",
         "SAP_OrderLineItem.SAP_OrderLineObjectIdentifier.ItemId",
               "Item",
               "SAPMbasc",
               strInitiator);

            throw new MapFailureException("ForeignKeyXref() failed");
            }
         }
      }
   }
```

See also

foreignKeyLookup()

"Performing foreign key lookups" on page 216

maintainChildVerb()

Sets the child business object verb based on the map execution context and the verb of the parent business object.

Syntax

```
public static void maintainChildVerb (String relDefName,
      String appSpecificParticpntName,
String genericParticpntName,
      BusObj appSpecificObj,
String appSpecificChildObj,
      BusObj genericObj,
String genericChildObj,
      CxExecutionContext map_ctx,
boolean to_Retrieve,
boolean is_Composite)
```

Parameters

relDefName — The name of the identity relationship name that manages the child business object.

appSpecificParticpntName — The name of the application-specific participant definition.

genericParticpntName — The name of the generic participant definition.

appSpecificObj — The application-specific object that contains the child object.

appSpecificChildObj — The name of the application child business object.

genericObj — The generic business object to or from which the *appSpecificObject* is being mapped.

genericChildObj — The name of the generic child business object.

ctx — The execution context.

to_Retrieve — The flag for the SERVICE_CALL_RESPONSE logic. When the condition is true, update the verbs of the child business objects. If false, do nothing.

isComposite — The flag that indicates whether the child participant uses composite keys. If the condition is true, keys are used; if false, keys are not used.

Return values

None.

Exceptions

`RelationshipRuntimeException`—see the Notes section for more information on when this exception is thrown

`ClassCastException`

Notes

The `maintainChildVerb()` method performs the following validations on arguments that are passed in:

- Validate the name of the *relDefName* relationship definition.
- Validate the name of the participant definitions for the application-specific business object (*appSpecificParticpntName*) and the generic business object (*genericParticpntName*).
- Make sure that the application-specific (*appSpecificObject*) and generic business objects (*genericObject*) are *not* null.
- Make sure that the *relDefName* relationship is an identity relationship. In addition, the participant definition in *relDefName* that represents the generic business object must be defined as `IBM WebSphere InterChange Server Express-managed`. For more information on how to specify these settings, see "Defining identity relationships" on page 174.

If any of these validations fails, `maintainChildVerb()` throws the `RelationshipRuntimeException` exception.

Once the arguments are validated, the action that `maintainChildVerb()` takes depends on the following information:

- The calling context—in the map execution context, passed in as part of the *map_ctx* argument (`cwExecCtx`)
- The verb—in the source business object
  - Application-specific business object (*appSpecificObj*) for calling contexts `EVENT_DELIVERY` (or `ACCESS_REQUEST`) and `SERVICE_CALL_RESPONSE`
  - Generic business object (*genericObj*) for calling context `SERVICE_CALL_REQUEST`

For more information on the actions that `maintainChildVerb()` takes, see "Determining the child verb setting" on page 213. Table 84 through Table 87 provide the actions for each of the calling contexts.

You call this method in the transformation step for the child attribute of a parent object. This child object can participant in either

- In the transformation step for the key attribute of a submap that transforms child business objects if the child business objects are related using a unique key.

You usually use `maintainChildVerb()` to set the verb of a child object that participates in a composite identity relationship (`maintainCompositeRelationship()`). However, you can also call it to set the verb of a child object that participates in a simple identity relationship (`maintainSimpleIdentityRelationship()`).

Examples

For an example involving `maintainChildVerb()`, see "Customizing map rules for a composite identity relationship" on page 204.

See also

`maintainCompositeRelationship()`, `maintainSimpleIdentityRelationship()`

"Setting the source child verb" on page 213

maintainCompositeRelationship()

Maintains a composite identity relationship from within the parent map.

Syntax

```
public static void maintainCompositeRelationship(String relDefName,
      String particpntDefName, BusObj appSpecificBusObj,
      Object genericBusObjList, CxExecutionContext map_ctx)
```

Parameters

relDefName
: The name of the composite identity relationship (as defined in Relationship Designer Express) in which the parent attribute participates.

particpntDefName
: The name of the participant that includes the composite key. This participant is always application-specific.

appSpecificBusObj
: The variable that contains the application-specific business object used in this map. This business object is the parent business object.

genericBusObjList
: The variable that contains the generic business object or objects used in this map, each generic business object is a contained child business object of the generic parent object. This parameter can be either a single generic business object (BusObj) or an array of generic business objects (BusObjArray).

map_ctx
: The map execution context. To pass the map execution context, use the cwExecCtx variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

RelationshipRuntimeException

CxMissingIDException
: If a participant does not exist in the relationship tables during a map execution with a verb of Retrieve and an calling context of SERVICE_CALL_REQUEST. The connector sends a "service call request failed" message to the collaboration without sending the business object to the application.

Notes

The maintainCompositeRelationship() method maintains the relationship table associated with the *particpntDefName* participant of the *relDefName* composite identity relationship. This method maintains a relationship whose participant uses keys from multiple business objects at different levels (a composite key).

Note: The maintainCompositeRelationship() method *cannot* handle the case where the child's composite key depends on its grandparents. For more information, see "Actions of General/APIs/Identity Relationship/Maintain Composite Relationship" on page 203.

This method iterates through all the child business objects in the *appSpecificObj* parent business object, maintaining the relationship instances in the *partDefName* participant's relationship table. The method obtains the relationship instance IDs from the array of generic business objects that it receives (*genericObjs*). For each child instance, maintainCompositeRelationship() calls the maintainSimpleIdentityRelationship() method to perform the actual relationship-table management. The action that maintainSimpleIdentityRelationship() takes depends on the following information:

- The calling context—in the map execution context, passed in as part of the *map_ctx* argument (cwExecCtx)
- The verb—in the source business object, which is either:
  - Application-specific business object (*appSpecificBusObj*) for calling contexts EVENT_DELIVERY (or ACCESS_REQUEST) and SERVICE_CALL_RESPONSE
  - Generic business object (one element of the *genericBusObjList* array) for calling contexts SERVICE_CALL_REQUEST and ACCESS_RESPONSE

For more information on the actions that `maintainSimpleIdentityRelationship()` takes, see "Accessing identity relationship tables" on page 191. Table 74 through Table 78 provide the actions for each of the calling contexts.

Use `maintainCompositeRelationship()` in conjunction with the `maintainChildVerb()` and `updateMyChildren()` methods to maintain a composite relationship. For more information, see "Customizing map rules for a composite identity relationship" on page 204.

Examples

```
// This is an example of a code fragment in a parent map. It maintains
// the relationship table for all instances of a child object type for
// this application-specific parent object.

BusObjArray secondLevel2 =
      (BusObjArray)ObjFirstLevelBusObj2.get("MultiCardChild");

IdentityRelationship.maintainCompositeRelationship(
   "CmposRel",
   "AppSpPrt",
   ObjFirstLevelBusObj2,
   secondLevel2,
   cwExecCtx);

IdentityRelationship.updateMyChildren(
   "PCRel",
   "Parent",
   ObjFirstLevelBusObj2,
   "Child",
   "MultiCardChild",
   "CmposRel",
   "AppSpPrt",
   cwExecCtx);
```

For more examples involving `maintainCompositeRelationship()`, see "Customizing map rules for a composite identity relationship" on page 204.

See also

`updateMyChildren()`, `maintainChildVerb()`, `maintainSimpleIdentityRelationship()`

"Using composite identity relationships" on page 202

---

maintainSimpleIdentityRelationship()

Maintains a simple identity relationship from within either a parent or child map.

Syntax

```
public static void maintainSimpleIdentityRelationship(
      String relDefName, String particpntDefName,
      BusObj appSpecificBusObj, BusObj genericBusObj,
      CxExecutionContext map_ctx)
```

Parameters

relDefName	The name of the simple identity relationship (as defined in Relationship Designer Express) in which this attribute participates.

particpntDefName
: The name of the participant definition that represents the application-specific business object.

appSpecificBusObj
: The variable that contains the application-specific business object used in this map.

genericBusObj
: The variable that contains the generic business object used in this map.

map_ctx
: The map execution context. To pass the map execution context, use the `cwExecCtx` variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

`RelationshipRuntimeException`
: see the Notes section for more information on when this exception is thrown.

`CxMissingIDException`
: If a participant does not exist in the relationship tables during a map execution with a verb of Retrieve and an calling context of `SERVICE_CALL_REQUEST`. The connector sends a "service call request failed" message to the collaboration without sending the business object to the application.

Notes

The `maintainSimpleIdentityRelationship()` method maintains the relationship table associated with the *particpntDefName* participant of the *relDefName* simple identity relationship. This method maintains a relationship whose participant uses unique keys from multiple business objects at the same level.

The `maintainSimpleIdentityRelationship()` method performs the following validations on arguments that are passed in:

- Validate the name of the *relDefName* relationship definition.
- Validate the name of the *particpntDefName* participant definition for the application-specific business object.
- Make sure that the application-specific (*appSpecificBusObj*) and generic business objects (*genericBusObj*) are *not* null.
- Make sure that the *relDefName* relationship is an identity relationship. In addition, the participant definition in *relDefName* that represents the generic business object must be defined as `IBM WebSphere InterChange Server Express-managed`. For more information on how to specify these settings, see "Defining identity relationships" on page 174.
- Make sure the calling context is valid (see Table 73 for a list of valid calling contexts).
- Make sure that the application-specific business object's verb is supported. It must be one of the following: Create, Update, Delete, Retrieve.

If any of these validations fails, `maintainSimpleIdentityRelationship()` throws the `RelationshipRuntimeException` exception.

Once the arguments are validated, the action that `maintainSimpleIdentityRelationship()` takes depends on the following information:

- The calling context—in the map execution context, passed in as part of the *map_ctx* argument (`cwExecCtx`)
- The verb—in the source business object
  - Application-specific business object (*appSpecificBusObj*) for calling contexts `EVENT_DELIVERY` (or `ACCESS_REQUEST`) and `SERVICE_CALL_RESPONSE`
  - Generic business object (*genericBusObj*) for calling contexts `SERVICE_CALL_REQUEST` and `ACCESS_RESPONSE`

The `maintainSimpleIdentityRelationship()` method handles all of the basic adding and deleting of participants and relationship instances for each combination of calling context and verb. For more information on the actions that `maintainSimpleIdentityRelationship()` takes, see "Accessing identity relationship tables" on page 191. Table 74 through Table 78 provide the actions for each of the calling contexts.

You can call this method in either of the following cases:

- In the transformation step for the key attribute of a parent object
- In the transformation step for the key attribute of a submap that transforms child business objects if the child business objects are related using a unique key.

Use `maintainSimpleIdentityRelationship()` in conjunction with the `maintainChildVerb()` method to maintain a simple identity relationship. For more information, see "Defining transformation rules for a simple identity relationship" on page 201.

Examples

The following example maintains the simple identity relationship between the `Clarify_BusOrg` and generic `Customer` business objects in an inbound `Clarify_BusOrg`-to-`Customer` map:

```
IdentityRelationship.maintainSimpleIdentityRelationship(
   "CustIdentity",
   "ClarBusOrg",
   ObjClarify_BusOrg,
   ObjCustomer,
   cxExecCtx);
```

For more examples involving `maintainSimpleIdentityRelationship()`, see "Defining transformation rules for a simple identity relationship" on page 201.

See also

`maintainChildVerb()`

"Using simple identity relationships" on page 191

updateMyChildren()

Adds and deletes child instances in a specified parent/child relationship of an identity relationship as necessary.

Syntax

```
void updateMyChildren(String parentChildRelDefName,
      String parentParticpntDef, BusObj parentBusObj,
      String childParticpntDef, String childAttrName,
      String childIdentityRelDefName,
      String childIdentityParticpntDefName,
      CxExecutionContext map_ctx)
```

Parameters

parentChildRelDefName
: The name of the parent/child relationship definition.

parentParticpntDefName
: The name of the participant definition that represents the parent business object in the parent/child relationship.

parentBusObj
: The variable that contains the parent business object.

childParticpntDefName
: The name of the participant definition that represents the child business object in the parent/child relationship.

childAttrName
: The name of the attribute in the parent business object whose type is the child object name that participates in the parent/child relationship. For example, in a customer-address relationship, if the parent object contains an `Address1` attribute, which is a child business object of type `Address`, the *childAttrName* attribute name is `Address1`.

childIdentityRelDefName
: The name of the identity relationship in which the child business object participates.

childIdentityParticpntDefName
: The name of the participant definition that represents the child business object in the identity relationship.

map_ctx
: The map execution context. To pass the map execution context, use the `cwExecCtx` variable, which Map Designer Express defines for every map.

Return values

None.

Exceptions

`RelationshipRuntimeException`
: see the Notes section for more information on when this exception is thrown

Notes

The `updateMyChildren()` method updates the child instances in the relationship tables of the *parentChildRelDefName* and *childIdentityRelDefName* relationship definitions. This method is useful in an identity relationship when a parent business object has been updated as a result of the addition or removal of child objects. Use `updateMyChildren()` to compare the after-image (in *parentBusObj*) with the before-image (information in the relationship tables) to determine which child objects in the after-image are new or deleted.

Note: The `updateMyChildren()` method *cannot* handle the case where the child's composite key depends on its grandparents. For more information, see "Tips on using Update My Children" on page 209.

The `updateMyChildren()` method performs the following validations on arguments that are passed in:

- Validate the name of the *parentChildrelDefName* relationship definition (first argument).
- Make sure that the *parentChildRelDefName* relationship is a parent/child relationship and that the *parentParticpntDefName* and *childParticpntDefName* are part of the *parentChildRefDefName* relationship definition.
- Make sure that the *childIdentityRelDefName* relationship is an identity relationship. In addition, the participant definition in *childIdentityRelDefName* that represents the generic business object must be defined as `IBM WebSphere InterChange Server Express- managed`. For more information on how to specify these settings, see "Defining identity relationships" on page 174.
- Make sure that the *childIdentityParticpntDefName* is part of the *childIdentityRefDefName* relationship definition

If any of these validations fails, `updateMyChildren()` throws the `RelationshipRuntimeException` exception.

Once the arguments are validated, the `updateMyChildren()` method adds children or deletes children from the list of child business objects that belong to the specified parent business object as appropriate. This method performs one of the following tasks to the relationship tables for the parent and child participants (*parentParticpntDefName* and *childParticpntDefName*, respectively):

- For each new child object, `updateMyChildren()` adds a child instance.

  This method does *not* add to the child's relationship table because all the business objects that are currently associated with the parent object have already been maintained when `maintainCompositeRelationship()` was called.
- For each deleted child object, `updateMyChildren()` removes the corresponding child instance.

  This method removes from the child's cross-reference table in addition to the parent/child relationship table.

The `updateMyChildren()` method requires that a parent/child relationship is defined with Relationship Designer Express. For information on how to create this kind of relationship, see "Creating the parent/child relationship definition" on page 208.

Note: If the child business object has a unique key, the child participant's attribute is the unique key of the child object. If the child object does not have a unique key, the child participant's attribute is this nonunique key.

Examples

For an example involving `updateMyChildren()` in conjunction with the `maintainCompositeRelationship()` method, see the Examples section of `maintainCompositeRelationship()`.

For more examples involving `updateMyChildren()`, see "Customizing map rules for a composite identity relationship" on page 204.

See also

addMyChildren(), deleteMyChildren(), maintainCompositeRelationship(), maintainSimpleIdentityRelationship()

"Handling updates to the parent business object" on page 209

Chapter 21. MapExeContext class

The `MapExeContext` class provides methods for querying and setting various runtime values that are in effect during map execution.

Table 106 summarizes the methods of the `MapExeContext` class.

Table 106. MapExeContext method summary

Method	Description	Page
`getConnName()`	Retrieves the connector name associated with the current map instance.	363
`getInitiator()`	Retrieves the calling context associated with the current map instance.	363
`getLocale()`	Retrieves the locale associated with the map execution context.	364
`getOriginalRequestBO()`	Retrieves the original-request business object associated with the current map instance.	365
`setConnName()`	Sets the connector name associated with the current map instance.	366
`setInitiator()`	Sets the calling context associated with the current map instance.	366
`setLocale()`	Sets the locale associated with the map execution context.	366

getConnName()

Retrieves the connector name associated with the current map instance.

Syntax

```
String getConnName()
```

Parameters

None.

Return values

Returns a String containing the connector name.

Exceptions

None.

See also

`setConnName()`

getInitiator()

Retrieves the calling context associated with the current map instance.

Syntax

```
String getInitiator()
```

© Copyright IBM Corp. 2003

Parameters

None.

Return values

Returns a static constant variable representing the calling context for the execution of the current map instance. Calling contexts are one of the following values:

`EVENT_DELIVERY`
: The source business objects being mapped are sent from an application to InterChange Server Express through a connector.

`ACCESS_REQUEST`
: The source objects being mapped are sent from an application to InterChange Server Express through an access client.

`SERVICE_CALL_REQUEST`
: The source objects being mapped are sent from InterChange Server Express to an application through a connector.

`SERVICE_CALL_RESPONSE`
: The source objects being mapped are sent back to InterChange Server Express from an application through a connector after a successful service call request.

`SERVICE_CALL_FAILURE`
: The source objects being mapped are sent back to InterChange Server Express from an application through a connector after a failed service call request.

`ACCESS_RESPONSE`
: The source objects being mapped are sent back from InterChange Server Express to the application through an access client.

Exceptions

None.

Notes

The calling context is part of the map execution context. For more information on how calling contexts are used in maps, see "Understanding map execution contexts" on page 146.

Examples

In the following example, compare the map run-time initiator with the constants defined in the MapExeContext class:

```
String sInitiator = null;
sInitiator = cwMapCtx.getInitiator();
if(sInitiator.equals(MapExeContext.EVENT_DELIVERY))
     logInfo("**************Initiator =  MapExeContext.EVENT_DELIVERY.");
```

See also

`getOriginalRequestBO()`, `setInitiator()`

getLocale()

Retrieves the locale associated with the map execution context.

Syntax

```
Locale getLocale()
```

Parameters

None.

Return values

Returns a Locale object that contains the language and country code for the map execution context.

Exceptions

None.

Notes

This method must be run on the map variable of `MapExeContext` type, which is named `cwMapCtx` when generated by the system, or which you name when calling a map in an environment that does not automatically generate map code (such as within a collaboration).

Examples

The following example retrieves the locale of the map execution context into a variable and then reports it with a trace statement:

```
Locale mapLocale = cwMapCtx.getLocale();
String mapLocaleToString = mapLocale.toString();
trace(3, "THE MAP LOCALE IS: " + mapLocaleToString);
```

See also

`setLocale()`

getOriginalRequestBO()

Retrieves the original-request business object associated with the current map instance.

Syntax

```
BusObj getOriginalRequestBO()
```

Parameters

None.

Return values

Returns the original-request business object for the map, as the following table shows:

Calling Contexts	Original-Request Business Object
EVENT_DELIVERY, ACCESS_REQUEST	Application-specific business object that came in from the application
SERVICE_CALL_REQUEST, SERVICE_CALL_FAILURE	Generic business object that was sent down from InterChange Server Express
SERVICE_CALL_RESPONSE	Generic business object that was sent down by the SERVICE_CALL_REQUEST

Calling Contexts	Original-Request Business Object
ACCESS_RESPONSE	Application-specific business object that came in from the access request initially

Exceptions

None.

Notes

The original-request business object is part of the map execution context. The `getOriginalRequestBO()` method returns the original-request business object, which depends on the map's calling context. For more information on how this business object is used in maps, see "Original-request business objects" on page 148..

See also

`getInitiator()`

setConnName()

Sets the connector name associated with the current map instance.

Syntax

```
void setConnName(String connectorName)
```

Parameters

connectorName Name of the connector

Return values

None.

Exceptions

None.

Notes

The controller for the connector you specify must be running in InterChange Server Express.

See also

`getConnName()`

setInitiator()

Sets the calling context associated with the current map instance.

Syntax

```
void setInitiator(String callingContext)
```

Parameters

callingContext

String containing one of the following values:

EVENT_DELIVERY	The source objects being mapped are sent from an application through a connector to InterChange Server Express.
ACCESS_REQUEST	The source objects being mapped are sent from an application to InterChange Server Express through an access client.
SERVICE_CALL_REQUEST	The source objects being mapped are sent from InterChange Server Express to an application through a connector.
SERVICE_CALL_RESPONSE	The source objects being mapped are sent back to InterChange Server Express from an application through a connector after a successful service call request.
SERVICE_CALL_FAILURE	The source objects being mapped are sent back to InterChange Server Express from an application through a connector after a failed service call request.
ACCESS_RESPONSE	The source objects being mapped are sent back from InterChange Server Express to the application through an access client.

Return values

None.

Exceptions

None.

Notes

The calling context is part of the map execution context. The calling context indicates the direction in which the source business object is being mapped. For more information on how calling contexts are used in maps, see "Understanding map execution contexts" on page 146. .

See also

`getInitiator()`

---

setLocale()

Sets the locale associated with the map execution context.

Syntax

```
void setLocale(Locale newLocale)
```

Parameters

newLocale The new `Locale` object to set the map execution context to.

Return values

None.

Exceptions

None.

Notes

This method must be run on the map variable of `MapExeContext` type, which is named `cwMapCtx` when generated by the system, or which you name when calling a map in an environment that does not automatically generate map code (such as within a collaboration).

The locale of the business object produced by a map is affected by the local of the map's execution context. If you change the locale of the map execution context as part of the map's logic, therefore, the new locale is copied to the business object. This is done when the user-modifiable logic is finished executing (that is, when the transformations visible in the diagram of the Map Designer Express are finished). You can use this API to change the business object to a different locale than the one it had when it entered the map.

Examples

The code below defines a new `Locale` object, sets the map execution context to that new Locale value, and then reports the map execution context locale:

```
Locale newLocale = new Locale("ja", "JP");
cwMapCtx.setLocale(newLocale);
trace(3, "THE MAP LOCALE IS NOW: " + cwMapCtx.getLocale().toString());
```

See also

`getLocale()`

Deprecated methods

Some methods in the `MapExeContext` class were supported in earlier versions but are no longer supported. These *deprecated methods* will not generate errors, but CrossWorlds recommends that you avoid their use and migrate existing code to the new methods. The deprecated methods might be removed in a future release.

Table 107 lists the deprecated method for the `MapExeContext` class. If you have not used Map Designer Express before, ignore this section.

Table 107. Deprecated Method, MapExeContext Class

Former method	**Replacement**
getGenericBO()	getOriginalRequestBO()

Chapter 22. Participant class

The methods documented in this chapter operate on objects of the `Participant` class. Participant instances are used in relationship instances. Each Participant instance contains the following information:

- name of the relationship definition
- relationship instance ID
- name of the participant definition
- data to associate with the participant

The `Participant` class provides methods for setting and retrieving each of these values for a given participant.

Table 108 summarizes the methods of the `Participant` class.

Table 108. Participant method summary

Method	Description	Page
`Participant()`	Creates a new `Participant` instance.	369
`getBusObj(), getString(), getLong(), getInt(), getDouble(), getFloat(), getBoolean()`	Retrieves the data associated with the `Participant` instance.	371
`getInstanceId()`	Retrieves the relationship instance ID of the relationship in which the `Participant` instance is participating.	371
`getParticipantDefinition()`	Retrieves the participant definition name from which the `Participant` instance is created.	372
`getRelationshipDefinition()`	Retrieves the name of the relationship definition in which the `Participant` instance is participating.	372
`set()`	Sets the data associated with the `Participant` instance.	373
`setInstanceId()`	Sets the instance ID of the relationship in which the `Participant` instance is participating.	373
`setParticipantDefinition()`	Sets the participant definition name from which the `Participant` instance is created.	374
`setRelationshipDefinition()`	Sets the relationship definition in which the `Participant` instance is participating.	374

Participant()

Creates a new `Participant` instance.

© Copyright IBM Corp. 2003

Syntax

To add a new participant instance to an existing participant in a relationship instance:

```
Participant(String relDefName,String partDefName, int instanceId,BusObj partData)
Participant(String relDefName,String partDefName, int instanceId,String partData)
Participant(String relDefName,String partDefName, int instanceId,long partData)
Participant(String relDefName,String partDefName, int instanceId,int partData)
Participant(String relDefName,String partDefName, int instanceId,double partData)
Participant(String relDefName,String partDefName, int instanceId,float partData)
Participant(String relDefName,String partDefName, int instanceId,boolean partData)
```

To create a new participant instance with no relationship instance:

```
Participant(String relDefName,String partDefName, BusObj partData)
Participant(String relDefName,String partDefName, String partData)
Participant(String relDefName,String partDefName, long partData)
Participant(String relDefName,String partDefName, int partData)
Participant(String relDefName,String partDefName, double partData)
Participant(String relDefName,String partDefName, float partData)
Participant(String relDefName,String partDefName, boolean partData)
```

Parameters

relDefName
Name of the relationship definition.

partDefName
Name of the participant definition that describes the participant.

instanceId
The relationship instance ID for the relationship instance to receive the new participant instance.

participantData
Data to associate with the participant instance. Can be one of the following data types: `BusObj`, `String`, `long`, `int`, `double`, `float`, `boolean`.

Return values

Returns new participant instance.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

Notes

This method is the `Participant` class constructor. It takes the following forms:

- The first form of the constructor adds a new participant instance to the relationship instance identified by *instanceId*.
- The second form creates a new participant instance with no associated relationship instance. You can use this participant instance as an argument to `IdentityRelationship.addMyChildren()` or `Relationship.create()` to create a new relationship instance. With the `Relationship.create()` method, having no relationship instance ID is a requirement.

The data to associate with the *participantData* parameter depends on the kind of relationship:

- To create a participant instance for an identity relationship, use a business object as the *participantData* parameter.

- To create a participant for a lookup relationship, use any of the following data types for the *participantData* parameter: String, long, int, double, float, boolean.

Examples

```
// create a participant instance with no relationship instance ID
participant p = new Participant(myRelDef,myPartDef,myBusObj);

// create a relationship instance
int relInstanceId = Relationship.addParticipant(p);
```

See also

addMyChildren(), Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

getBusObj(), getString(), getLong(), getInt(), getDouble(), getFloat(), getBoolean()

Retrieves the data associated with the Participant instance.

Syntax

```
BusObj getBusObj()
String getString()
long getLong()
int getInt()
double getDouble()
float getFloat()
boolean getBoolean()
```

Return values

Returns the data associated with this participant instance. This data value is of the type included in the method name. For example, getBoolean() returns a boolean value, getBusObj() returns a BusObj value, getDouble() returns a double value, and so on.

Exceptions

RelationshipRuntimeException – See "Handling exceptions" on page 144.

See also

set(), Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

getInstanceId()

Retrieves the relationship instance ID of the relationship in which the Participant instance is participating.

Syntax

```
int getInstanceId()
```

Return values

Returns an integer representing the instance ID of the relationship instance in which this `Participant` instance is participating. If the `Participant` instance is *not* a member of a relationship instance, this method returns the constant, `INVALID_INSTANCE_ID`.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

See also

`setInstanceId()`, Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

getParticipantDefinition()

Retrieves the participant definition name from which the `Participant` instance is created.

Syntax

```
String getParticipantDefinition()
```

Return values

Returns a `String` containing the name of the participant definition associated with this participant instance.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

See also

`setParticipantDefinition()`, Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

getRelationshipDefinition()

Retrieves the name of the relationship definition in which the `Participant` instance is participating.

Syntax

```
String getRelationshipDefinition()
```

Return values

Returns a `String` containing the name of the relationship definition in which this participant instance participates.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

See also

`setRelationshipDefinition()`, Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

set()

Sets the data associated with the `Participant` instance.

Syntax

```
void set(BusObj partData)
void set(String partData)
void set(long partData)
void set(int partData)
void set(double partData)
void set(float partData)
void set(boolean partData)
```

Parameters

partData	Data to associate with the `Participant` instance. Can be one of the following data types: BusObj, String, long, int, double, float, boolean.

Return values

None.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

Notes

If you set the participant data to be a business object (`BusObj` type), the relationship definition and participant definition must already be set. If you set the participant data to any other data type, it does not matter which setting you specify first.

See also

`getBusObj()`, `getString()`, `getLong()`, `getInt()`, `getDouble()`, `getFloat()`, `getBoolean()`, Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

setInstanceId()

Sets the instance ID of the relationship in which the `Participant` instance is participating.

Syntax

```
void setInstanceId(int id)
```

Parameters

id	Instance ID of the relationship.

Return values

None.

Exceptions

RelationshipRuntimeException – See “Handling exceptions” on page 144.

Notes

One use of setInstanceId() is to remove the relationship instance ID when you want to pass a participant instance as a parameter to the Participant() or create() methods. In this case, you set the instance ID to the constant INVALID_INSTANCE_ID.

Examples

```
// wipe out the relationship instance ID
myParticipant.setInstanceId(Participant.INVALID_INSTANCE_ID);

// pass the participant instance to the create() method
int newRelId = create(myParticipant);
```

See also

getInstanceId(), Chapter 7, “Creating relationship definitions,” on page 167, “Transforming with a submap” on page 41

---

setParticipantDefinition()

Sets the participant definition name from which the Participant instance is created.

Syntax

```
void setParticipantDefinition(String partDefName)
```

Parameters

partDefName Name of the participant definition from which the Participant instance is created.

Return values

None.

Exceptions

RelationshipRuntimeException – See “Handling exceptions” on page 144.

See also

setParticipantDefinition(), Chapter 7, “Creating relationship definitions,” on page 167, “Transforming with a submap” on page 41

---

setRelationshipDefinition()

Sets the relationship definition in which the Participant instance is participating.

Syntax

```
void setRelationshipDefinition(String relDefName)
```

Parameters

relDefName Name of the relationship definition.

Return values

None.

Exceptions

RelationshipRuntimeException – See "Handling exceptions" on page 144.

See also

getRelationshipDefinition(), Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

Chapter 23. Relationship class

The methods documented in this chapter operate on objects of the IBM WebSphere InterChange Server Express-defined class `Relationship`. The `Relationship` class provides methods for manipulating the runtime instances of relationships, called *relationship instances*. You typically use these methods in transformation steps for business object attributes that are mapped as identity relationships or static lookups. For more information on programming relationship attributes using the methods in this class, see “Transforming with a submap” on page 41.

Most methods in this class support variations in the parameters you specify. The variations generally follow these guidelines:

- To identify a specific participant in a relationship instance, you usually specify the relationship definition name, the participant definition name, the relationship instance ID, and the business object associated with the participant.
- Alternatively, you can specify a Participant instance which contains the relationship definition name, participant definition name, instance ID and business object, as its attributes.
- For some operations, you can omit the relationship instance ID (for example, when creating a new relationship) or the business object name.

In most cases, if you have a Participant instance (for example, as the result of a `retrieve()` call), it is easier to pass it as a parameter to a `Relationship` class method instead of specifying each attribute individually.

All methods in this class are declared as static. You can call them from existing relationship instances or by referencing the `Relationship` class.

Table 109 summarizes the methods in the `Relationship` class.

Table 109. Relationship method summary

Method	Description	Page
Static methods		
`addParticipant()`	Adds a new participant to a relationship instance.	378
`create()`	Creates a new relationship instance.	380
`deactivateParticipant()`	Deactivates a participant from one or more relationship instances.	381
`deactivateParticipantByInstance()`	Deactivates a participant from a specific relationship instance.	382
`deleteParticipant()`	Removes a participant instance from one or more relationship instances.	383
`deleteParticipantByInstance()`	Removes a participant from a specific relationship instance.	384
`getNewID()`	Returns the next available relationship instance ID for a relationship, based on the relationship definition name.	385
`retrieveInstances()`	Retrieves only the relationship instance IDs for zero or more relationship instances which contain a given participant instance.	386
`retrieveParticipants()`	Retrieves zero or more participants from a relationship instance.	388

© Copyright IBM Corp. 2003

Table 109. Relationship method summary (continued)

Method	Description	Page
updateParticipant()	Updates a participant in one or more relationship instances.	389
updateParticipantByInstance()	Updates a participant in a specific relationship instance.	389

addParticipant()

Adds a new participant to a relationship instance.

Syntax

To add a new participant to an existing relationship instance:

```
int addParticipant
 (String relDefName,
String partDefName,
int instanceId,BusObj partData)

int addParticipant
 (String relDefName,
String partDefName,
 int instanceId,String partData)

int addParticipant
 (String relDefName,
String partDefName,int instanceId,
long partData)

int addParticipant
 (String relDefName,
String partDefName,int instanceId,
int partData)

int addParticipant
 (String relDefName,
String partDefName,
int instanceId,
double partData)

int addParticipant
 (String relDefName,
String partDefName,
int instanceId, float partData)

int addParticipant
 (String relDefName,
String partDefName,
int instanceId,
boolean partData)
```

To add a participant to a new relationship instance:

```
int addParticipant
 (String relDefName,
String partDefName,
      BusObj partData)
int addParticipant
 (String relDefName,
String partDefName,
      String partData)
int addParticipant
 (String relDefName,
```

```
String partDefName,
      long partData)
int addParticipant
 (String relDefName,
String partDefName,
      int partData)
int addParticipant
 (String relDefName,
String partDefName,
      double partData)
int addParticipant
 (String relDefName,
String partDefName,
      float partData)
int addParticipant
 (String relDefName,
String partDefName,
      boolean partData)
```

To add an existing participant instance to a relationship instance:

```
int addParticipant(Participant participant)
```

Parameters

relDefName Name of the relationship definition.

partDefName Name of the participant definition.

instanceId Relationship instance ID of the relationship instance to receive the new participant.

partData Data to associate with the participant. Can be one of the following data types: `BusObj`, `String`, `long`, `int`, `double`, `float`, `boolean`.

participant Participant to add to the relationship.

Return values

Returns an integer representing the instance ID of the relationship to receive the new participant.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

Notes

The first form of the method adds a new participant to the relationship instance you specify. Each variation supports a different data type for the data to associate with the participant.

The second form, since it does not specify a relationship instance, creates a new relationship instance and adds the new participant. In this case, the return value is the instance ID of the newly created relationship instance. Each variation supports a different data type for the data to associate with the participant.

The third form adds the participant instance you pass to the relationship instance specified in the participant instance. If the participant instance has no relationship instance ID, a new relationship instance is created and the new instance ID is returned.

The addParticipant() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

create()

create()

Creates a new relationship instance.

Syntax

```
int create(String relDefName, String partDefName, BusObj partData)
int create(String relDefName, String partDefName, String partData)
int create(String relDefName, String partDefName, long partData)
int create(String relDefName, String partDefName, int partData)
int create(String relDefName, String partDefName, double partData)
int create(String relDefName, String partDefName, float partData)
int create(String relDefName, String partDefName, boolean partData)
int create(Participant participant)
```

Parameters

relDefName	Name of the relationship definition.
partDefName	The name of the participant definition.
partData	Data to associate with the participant. Can be one of the following data types: BusObj, String, long, int, double, float, boolean.
participant	First participant in the relationship.

Return values

Returns an integer representing the relationship instance ID of the new relationship.

Exceptions

RelationshipRuntimeException

Notes

The create() method creates a new relationship instance with one participant instance of the *partDefName* participant definition. You can specify the data for this new participant instance with the *partData* argument. After calling this method, you can call addMyChildren() to add more participants to the relationship instance.

In the last form of the method, the *participant* parameter cannot have a relationship instance ID. Normally, participant instances do have relationship instance IDs. Because this method creates a new relationship instance, you must make sure that the participant instance does not already have an instance associated with it. To do this, use the setInstanceId() method (in the Participant class) to set the instance ID to the INVALID_INSTANCE_ID constant.

The create() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

`addMyChildren()`, `setInstanceId()`

deactivateParticipant()

Deactivates a participant from one or more relationship instances.

Syntax

```
void deactivateParticipant(String relDefName,
String partDefName,
      BusObj partData)

void deactivateParticipant(String
relDefName,
String partDefName,
      String partData)

void deactivateParticipant(String relDefName,
String partDefName,
       long partData)

void deactivateParticipant(String relDefName,
String partDefName,
       int partData)

void deactivateParticipant(String relDefName,
String partDefName,
       double partData)

void deactivateParticipant(String relDefName,
String partDefName,
      float partData)

void deactivateParticipant(String relDefName,
String partDefName,
      boolean partData)

void deactivateParticipant(Participant participant)
```

Parameters

relDefName	Name of the relationship definition.
partDefName	Name of the participant definition.
partData	Data associated with the participant. Can be one of the following data types: `BusObj`, `String`, `long`, `int`, `double`, `float`, `boolean`.
participant	Participant to deactivate in the relationship.

Return values

None.

Exceptions

`RelationshipRuntimeException`

Notes

The `deactivateParticipant()` method deactivates the participant from all instances of *relDefName* where *partData* is associated with *partDefName*. This method does

not remove the participant from the relationship tables. Use this method to remove a participant while preserving a record of its existence in the relationship tables.

To view deactivated participants, you can query the relationship tables directly. To find the table names and access information for a given relationship, open the relationship definition using Relaitonship Designer Express and choose Advanced Settings from the Edit menu. See "Specifying advanced relationship settings" on page 179 for more information on these settings.

Attention: Because `deactivateParticipant()` does not actually remove participant rows from your relationship tables, you should not use this method routinely to delete participants. Doing so can cause your relationship tables to become unnecessarily large.

The `deactivateParticipant()` method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the `Relationship` class.

See also

`deleteParticipant()`, `deactivateParticipantByInstance()`, Chapter 7, "Creating relationship definitions," on page 167, "Transforming with a submap" on page 41

deactivateParticipantByInstance()

Deactivates a participant from a specific relationship instance.

Syntax

```
void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, BusObj partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, String partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, long partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, int partData ] )

void deactivateParticipantByInstance(String relDefName,
       String partDefName, int instanceId [, double partData ] )

void deactivateParticipantByInstance(String relDefName,
      String partDefName, int instanceId [, float partData ] )

void deactivateParticipantByInstance(String relDefName,
      String partDefName, int instanceId [, boolean partData ] )
```

Parameters

relDefName
: Name of the relationship definition.

partDefName
: Name of the participant definition.

instanceId
: ID of the relationship instance to which the participant belongs.

partData
: Data associated with the participant. Can be one of the following data types: BusObj, String, long, int, double, float, boolean. This is an optional parameter

Return values

None.

Exceptions

`RelationshipRuntimeException` – See "Handling exceptions" on page 144.

Notes

The `deactivateParticipantByInstance()` method deactivates the specified participant from the relationship instance that relationship instance ID *instanceID* identifies. However, the method does *not* remove the participant from the relationship tables. Use this method when you want to remove a participant while preserving a record of its existence in the relationship tables.

To view deactivated participants, you can query the relationship tables directly. To find the table names and access information for a given relationship, open the relationship definition using Relaitonship Designer Express and choose Advanced Settings from the Edit menu. See "Specifying advanced relationship settings" on page 179 for more information on these settings.

Attention: Since `deactivateParticipantByInstance()` does not actually remove participant rows from your relationship tables, you should not use this method routinely to delete participants. Doing so can cause your relationship tables to become unnecessarily large.

The `deactivateParticipantByInstance()` method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the `Relationship` class.

See also

`deleteParticipant()`, `deactivateParticipant()`

deleteParticipant()

Removes a participant instance from one or more relationship instances.

Syntax

```
void deleteParticipant(String relDefName, String partDefName,   BusObj partData)
void deleteParticipant(String relDefName, String partDefName,    String partData)
void deleteParticipant(String relDefName, String partDefName,    long partData)
void deleteParticipant(String relDefName, String partDefName,    int partData)
void deleteParticipant(String relDefName, String partDefName,    double partData)
void deleteParticipant(String relDefName, String partDefName,    float partData)
void deleteParticipant(String relDefName, String partDefName,    boolean partData)

void deleteParticipant(Participant participant)
```

Parameters

relDefName — Name of the relationship definition.

partDefName — Name of the participant definition.

partData — Data associated with the participant. Can be one of the following data types: BusObj, String, long, int, double, float, boolean.

participant A Participant instance representing the participant to remove from the relationship.

Return values

None.

Exceptions

RelationshipRuntimeException

Notes

The deleteParticipant() method deletes the specified participant from all instances of *relDefName* where *partData* is associated with *partDefName* and deletes it from the underlying relationship tables.

The deleteParticipant() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

deactivateParticipant(), deleteParticipantByInstance()

deleteParticipantByInstance()

Removes a participant from a specific relationship instance.

Syntax

```
void deleteParticipantByInstance(String relDefName,
    String partDefName, int instanceId [, BusObj partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, String partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, long partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, int partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, double partData] )

void deleteParticipantByInstance(String relDefName,

String partDefName, int instanceId [, float partData] )

void deleteParticipantByInstance(String relDefName,
    String partDefName, int instanceId [, boolean partData] )
```

Parameters

relDefName Name of the relationship definition.

partDefName Name of the participant definition.

instanceId ID of the relationship instance to which the participant belongs.

partData	Data associated with the participant. Can be one of the following data types: BusObj, String, long, int, double, float, boolean. This is an optional parameter.

Return values

None.

Exceptions

RelationshipRuntimeException

Notes

The deleteParticipantByInstance() method deletes a participant instance from the relationship identified by the *instanceId* relationship instance ID. The method removes the participant from the relationship instance and from the underlying relationship tables.

If you supply the optional *partData* parameter, deleteParticipantByInstance() deletes the participant instance *only* if *partData* is the data associated with the *partDefName* participant definition.

The last form of the method accepts a participant instance as the only parameter. The participant instance must contain the relationship definition name, participant definition name, and either the instance ID or the participant data.

The deleteParticipantByInstance() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

deactivateParticipant()

getNewID()

Returns the next available relationship instance ID for a relationship, based on the relationship definition name.

Syntax

```
public static int getNewID(String relDefName)
```

Parameters

relDefName	Name of the relationship definition.

Return values

Returns a relationship instance ID, based on the relationship definition name.

Exceptions

RelationshipRuntimeException

Notes

Because the relationship instance ID can be used as the generic ID for the typical IBM WebSphere InterChange Server Express identity relationships, this new ID can be used as the generic ID for generic-to-generic relationships.

retrieveInstances()

Retrieves only the relationship instance IDs for zero or more relationship instances which contain a given participant instance.

Syntax

```
int[] retrieveInstances(String relDefName,
String partDefName,
     BusObj partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      String partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      long partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      int partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      double partData)

int[] retrieveInstances(String relDefName,
String partDefName,
      float partData)

int[] retrieveInstances(String relDefName,
String partDefName,
     boolean partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      BusObj partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      String partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
       long partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      int partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
      double partData)

int[] retrieveInstances(String relDefName,
String[] partDefList,
       float partData)

int[] retrieveInstances(String relDefName,
```

```
String[] partDefList,
      boolean partData)

int[] retrieveInstances(String relDefName, BusObj partData)
int[] retrieveInstances(String relDefName, String partData)
int[] retrieveInstances(String relDefName, long partData)
int[] retrieveInstances(String relDefName, int partData)
int[] retrieveInstances(String relDefName, double partData)
int[] retrieveInstances(String relDefName, float partData)
int[] retrieveInstances(String relDefName, boolean partData)
```

Parameters

relDefName Name of the relationship definition.

partDefName Name of the participant definition.

partDefList List of participant definitions.

partData Data to associate with the participant. Can be one of the following data types: BusObj, String, long, int, double, float, boolean.

Return values

Returns an array of integers that are the instance IDs of relationships containing the participant.

Exceptions

RelationshipRuntimeException

Notes

The retrieveInstances() method implements a lookup relationship in an inbound map. It obtains the relationship instance IDs from the relationship table that are associated with the specified participant instances (*partDefList* and *partData* or only *partData*). The method retrieves *only* those attributes that are associated with the *relDefName* relationship definition. It does *not* fill in any of the other attributes in the business object. Attributes associated with the relationship definition typically are the key attributes and any others that you explicitly select. See Chapter 7, “Creating relationship definitions,” on page 167 for more information on relationship definitions.

If retrieveInstances() does not find a relationship instance for the specified data, it does *not* raise an exception. Absence of data in the relationship table does not mean that the lookup was performed improperly. If you want to raise an exception when retrieveInstances() does not find a value, you must check the value of the instance IDs that the method returns and explicitly raise a MapFailureException if the value is null.

The retrieveInstances() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

addMyChildren(), deactivateParticipant(), deleteParticipant(), retrieveParticipants()

“Customizing map transformations for a lookup relationship” on page 191

retrieveParticipants()

Retrieves zero or more participants from a relationship instance.

Syntax

```
Participant[] retrieveParticipants(String relDefName,
       String partDefName, int instanceId)|

Participant[] retrieveParticipants(String relDefName,
       String[] partDefList, int instanceId)

Participant[] retrieveParticipants(String relDefName,
        int instanceId)
```

Parameters

relDefName Name of the relationship definition.

partDefName Name of the participant definition.

partDefList List of participant definitions.

instanceId The relationship instance ID of the relationship instance to which the participant belongs.

Return values

Returns an array of Participant instances.

Exceptions

RelationshipRuntimeException

Notes

The retrieveParticipants() method implements a lookup relationship in an outbound map. It obtains the participant instances from the relationship table that are associated with the specified *instanceID* relationship instance ID. The method retrieves *only* those attributes that are associated with the *relDefName* relationship definition. It does *not* fill in any of the other attributes in the business object. Attributes associated with the relationship definition typically are the key attributes and any others that you explicitly select. See Chapter 7, "Creating relationship definitions," on page 167 for more information on relationship definitions.

If retrieveParticipants() raises the RelationshipRuntimeException if it receives a null-valued *instanceId*. If you are not guaranteed that the retrieveInstances() method has returned a matching instance ID, check the value of *instanceId* for a null value *before* the call to retrieveParticipants().

The retrieveParticipants() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

addMyChildren(), deactivateParticipant(), deleteParticipant(), retrieveInstances()

"Customizing map transformations for a lookup relationship" on page 191

updateParticipant()

Updates a participant in one or more relationship instances.

Syntax

```
void updateParticipant(String relDefName, String partDefName,       BusObj partData)
```

Parameters

relDefName Name of the relationship definition.

partDefName Name of the participant definition that participates in the *relDefName* relationship.

partData Data to associate with the participant. Can be one of the following data types: BusObj.

Return values

None.

Exceptions

RelationshipRuntimeException

Notes

The updateParticipant() method updates *partData* in instances of *relDefName* where *partData* is associated with *partDefName*. This method updates the non-key attributes of the business object that is associated with the specified participant. Only the attributes that are associated with the relationship definition are updated.

The updateParticipant() method updates all participant instances in the *relDefName* relationship that have:

- A participant definition of *partDefName*
- Key value(s) that matches the key value(s) of the *partData* business object

This method updates the non-key attributes of the participant instances with the values in the partData business object. Only the attributes that are associated with the relationship definition are updated.

To modify a key attribute or a participant type that is *not* a business object (such as String, long, int, double, float, or boolean), you must first delete the participant using deleteParticipant() or deactivateParticipant() and then add a new participant using addMyChildren().

The updateParticipant() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

deleteParticipant(), deactivateParticipant(), addMyChildren()

updateParticipantByInstance()

Updates a participant in a specific relationship instance.

Syntax

To update a participant in a specific relationship instance:

```
void updateParticipantByInstance(String relDefName,
      String partDefName, int instanceId [ , BusObj partData ] )

void updateParticipantByInstance(Participant participant)
```

Parameters

relDefName	Name of the relationship definition.
partDefName	Name of the participant definition.
instanceId	The relationship instance ID that identifies the relationship to which the participant belongs.
partData	Data to associate with the participant. Can be one of the following data types: BusObj. This parameter is optional.
participant	Participant to update in the relationship.

Return values

None.

Exceptions

RelationshipRuntimeException

Notes

The updateParticipantByInstance() method updates the non-key attributes of the business object associated with the specified participant. Only the attributes that are associated with the relationship definition are updated.

To modify a key attribute or a participant type that is not a business object (such as String, long, int, double, float, or boolean), you must first delete the participant using deleteParticipant() or deactivateParticipant() and then add a new participant using addMyChildren().

The updateParticipantByInstance() method is a class method declared as static. You can call this method from an existing relationship instance or by referencing the Relationship class.

See also

deleteParticipant(), deactivateParticipant(), addMyChildren()

Deprecated methods

Some methods in the Relationship class have been moved to the IdentityRelationship class. These *deprecated methods* will not generate errors, but CrossWorlds recommends that you avoid their use and migrate existing code to the new methods. The deprecated methods might be removed in a future release.

Table 110 lists the deprecated methods for the Relationship class.

Table 110. Deprecated methods, Relationship class

Former method	Replacement
addMyChildren()	addMyChildren() in the IdentityRelationship class
deleteMyChildren()	deleteMyChildren() in the IdentityRelationship class
maintainCompositeRelationship()	maintainCompositeRelationship() in the IdentityRelationship class
maintainSimpleIdentityRelationship()	maintainSimpleIdentity Relationship() in the IdentityRelationship class
updateMyChildren()	updateMyChildren() in the IdentityRelationship class

Chapter 24. UserStoredProcedureParam class

The `UserStoredProcedureParam` class provides methods for handling argument values to stored procedures, which you execute on the relationship database. A `UserStoredProcedureParam` object describes a single parameter for a stored procedure.

Important: The `UserStoredProcedureParam` class and its methods are supported for backward compatibility *only*. These *deprecated methods* will not generate errors, but you should avoid using them and migrate existing code to the new methods. The deprecated methods might be removed in a future release. In new map development, use the `CwDBStoredProcedureParam` class and its methods to provide arguments to a stored procedure.

Table 111 summarizes the methods in the `UserStoredProcedureParam` class.

Table 111. UserStoredProcedureParam method summary

Method	Description	Page
`UserStoredProcedureParam()`	Constructs a new instance of `UserStoredProcedureParam` that holds argument information for the parameter of a stored procedure.	393
`getParamDataTypeJavaObj()`	Retrieves the data type of this stored-procedure parameter as a Java `Object`, such as `Integer`, `Double`, or `String`.	394
`getParamDataTypeJDBC()`	Retrieves the data type of this stored-procedure parameter as an integer JDBC data type.	395
`getParamIndex()`	Retrieves the index position of this stored-procedure parameter.	395
`getParamIOType()`	Retrieves the in/out parameter type for this stored-procedure parameter.	396
`getParamName()`	Retrieves the name of this stored-procedure parameter.	397
`getParamValue()`	Retrieves the value of this stored-procedure parameter.	397
`setParamDataTypeJavaObj()`	Sets the data type as a Java `Object` for this stored-procedure parameter.	398
`setParamDataTypeJDBC()`	Sets the data type as a JDBC data type for this stored-procedure parameter.	398
`setParamIndex()`	Sets the index position of this stored-procedure parameter.	399
`setParamIOType()`	Sets the in/out parameter type of this stored-procedure parameter.	399
`setParamName()`	Sets the name of this stored-procedure parameter.	400
`setParamValue()`	Sets the value of this stored-procedure parameter.	400

UserStoredProcedureParam()

Constructs a new instance of `UserStoredProcedureParam` that holds argument information for the parameter of a stored procedure.

© Copyright IBM Corp. 2003

Syntax

```
UserStoredProcedureParam(int paramIndex, String paramType,
   Object paramValue, String ParamIOType, String paramName)
```

Parameters

paramIndex — The index position of the associated parameter in the declaration of the stored procedure. Index numbering begins with one (1).

paramType — The data type (as a Java `Object`) of the associated parameter.

paramValue — The argument value to send to the stored procedure.

ParamIOType — The in/out type of the associated parameter. Valid types are: "IN" - parameter value is *input only*. "INOUT" - parameter value is *input and output*. "OUT" - parameter value is *output only*.

paramName — The name of the argument, to be used in later statements that build the `Vector` array.

Return values

Returns a new `UserStoredProcedureParam` object to hold the argument information for the argument at position *argIndex* in the declaration of the stored procedure.

Exceptions

`DtpConnectionException` – If a parameter is invalid.

getParamDataTypeJavaObj()

Retrieves the data type of this stored-procedure parameter as a Java `Object`, such as `Integer`, `Double`, or `String`.

Syntax

```
String getParamDataTypeJavaObj()
```

Parameters

None.

Return values

Returns the data type of the associated `UserStoredProcedureParam` parameter as a Java `Object`.

Exceptions

None.

Notes

A Java `Object` is one of two representations of the parameter data type stored in the `UserStoredProcedureParam` object. Use `getParamDataTypeJavaObj()` to obtain the Java `Object` data type, you should work with the Java `Object` data type because:

- For IN (and INOUT) parameters, you *must* provide the parameter value as a Java `Object`. Therefore, providing the parameter data type as a Java `Object` is more consistent.

- The `execStoredProcedure()` method sends parameters in a `Vector` parameter array. The `Vector` object can contain only elements that are Java `Objects`.

See also

`getParamDataTypeJDBC()`, `setParamDataTypeJavaObj()`

getParamDataTypeJDBC()

Retrieves the data type of this stored-procedure parameter as an integer JDBC data type.

Syntax

```
int getParamDataTypeJDBC()
```

Parameters

None.

Return values

Returns the data type of the associated `UserStoredProcedureParam` parameter as a JDBC data type.

Exceptions

None.

Notes

The JDBC data type is one of two representations of the parameter data type stored in the `UserStoredProcedureParam` object. JDBC data types are integer values and include the following:

- `java.sql.Types.INTEGER`
- `java.sql.Types.VARCHAR`
- `java.sql.Types.DOUBLE`
- `java.sql.Types.DATE`

These data types are defined in `java.sql.Types`.

Recommendation: You should use the Java `Object` data type instead of the JDBC data type. However, the Mapping API uses the JDBC internally so you can obtain its value from the `UserStoredProcedureParam` object with `getParamDataTypeJDBC()`.

See also

`getParamDataTypeJavaObj()`, `setParamDataTypeJDBC()`

getParamIndex()

Retrieves the index position of this stored-procedure parameter.

Syntax

```
int getParamIndex()
```

Parameters

None.

Return values

Returns the index position of the associated `UserStoredProcedureParam` parameter.

Exceptions

None.

Notes

The index position of a stored-procedure parameter is its position in the parameter list of the stored-procedure declaration. The first parameter has an index position of one (1). The index position does *not* refer to literal parameters that might be supplied to the stored procedure.

See also

`setParamIndex()`

---

getParamIOType()

Retrieves the in/out parameter type for this stored-procedure parameter.

Syntax

```
String getParamIOType()
```

Parameters

None.

Return values

Returns the in/out type of the associated `UserStoredProcedureParam` parameter.

Exceptions

None.

Notes

The in/out parameter type indicates how the stored procedure uses the parameter. It can be the string representation of one of the following:

- IN parameter

  An IN parameter is *input only*; that is, the stored procedure accepts its value as input but does *not* use the parameter to return a value. The `getParamIOType()` returns the in/out parameter type as "IN".
- INOUT parameter

  An INOUT parameter is *input and output*; that is, the stored procedure accepts its value as input and also uses the parameter to return a value. The `getParamIOType()` returns the in/out parameter type as "INOUT".
- OUT parameter

  An OUT parameter is *output only*; that is, the stored procedure does *not* read its value as input but does use the parameter to return a value. The `getParamIOType()` returns the in/out parameter type as "OUT".

See also

`setParamIOType()`

getParamName()

Retrieves the name of this stored-procedure parameter.

Syntax

```
String getParamName()
```

Parameters

None.

Return values

Returns the name of the parameter from the associated `UserStoredProcedureParam` object.

Exceptions

None.

Notes

The name of the parameter is informational only. It is used only for error messages and debugging. The parameter name is not needed to access the stored-procedure parameter because stored procedures are accessed by their index position in the stored-procedure declaration.

See also

`setParamName()`

getParamValue()

Retrieves the value of this stored-procedure parameter.

Syntax

```
Object getParamValue()
```

Parameters

None.

Return values

Returns the value of the associated `UserStoredProcedureParam` parameter as a Java `Object`.

Exceptions

None.

Notes

The `getParamValue()` method returns the parameter value as a Java `Object` (such as `Integer`, `Double`, or `String`). If the value returned to an OUT parameter is the JDBC `NULL`, `getParamValue()` returns the `null` constant.

See also

`setParamValue()`

---

setParamDataTypeJavaObj()

Sets the data type as a Java `Object` for this stored-procedure parameter.

Syntax

```
void setParamDataTypeJavaObj(String paramDataType)
```

Parameters

`paramDataType` The data type of the parameter as a Java `Object`.

Exceptions

`DtpConnectionException` – If the input data type is not supported.

Notes

A Java `Object` is one of two representations of the parameter data type stored in the `UserStoredProcedureParam` object. Use `setParamDataTypeJavaObj()` to set the data type as a Java `Object`. You should work with the Java `Object` data type because:

- For IN (and INOUT) parameters, you *must* provide the parameter value as a Java `Object`. Therefore, providing the parameter data type as a Java `Object` is more consistent.
- The `execStoredProcedure()` method sends parameters in a `Vector` parameter array. The `Vector` object can contain only elements that are Java `Objects`.

See also

`getParamDataTypeJavaObj()`, `setParamDataTypeJDBC()`

---

setParamDataTypeJDBC()

Sets the data type as a JDBC data type for this stored-procedure parameter.

Syntax

```
void setParamDataTypeJDBC(int paramDataType)
```

Parameters

`paramDataType` The data type of the parameter as a JDBC type.

Exceptions

`DtpConnectionException` – If the input data type is not supported.

Notes

Every `UserStoredProcedureParam` object contains two representations of its data type: Java `Object` and JDBC data type. You should use the Java `Object` data type because:

- For IN (and INOUT) parameters, you *must* provide the parameter value as a Java `Object`. Therefore, providing the parameter data type as a Java `Object` is more consistent.
- The `execStoredProcedure()` method sends parameters in a `Vector` parameter array. The `Vector` object can contain only elements that are Java `Objects`.

See also

`getParamDataTypeJDBC()`, `setParamDataTypeJavaObj()`

setParamIndex()

Sets the index position of this stored-procedure parameter.

Syntax

```
void setParamIndex(int paramIndex)
```

Parameters

paramIndex	The index position of the stored-procedure parameter

Notes

The index position of a stored-procedure parameter is its position in the parameter list of the stored-procedure declaration. The first parameter has an index position of one (1). The index position does *not* refer to literal parameters that might be supplied to the stored procedure.

See also

`getParamIndex()`

setParamIOType()

Sets the in/out parameter type of this stored-procedure parameter.

Syntax

```
void setParamIOType(String paramIOType)
```

Parameters

paramIOType	The I/O type of the stored-procedure parameter

Notes

The in/out parameter type indicates how the stored procedure uses the parameter. It can be any of the following:

- IN parameter

  An IN parameter is *input only*; that is, the stored procedure accepts its value as input but does *not* use the parameter to return a value. For an IN parameter, set the in/out parameter type to “IN”.
- INOUT parameter

  An INOUT parameter is *input and output*; that is, the stored procedure accepts its value as input and also uses the parameter to return a value. For an INOUT parameter, set the in/out parameter type to “INOUT”.
- OUT parameter

An OUT parameter is *output only*; that is, the stored procedure does *not* read its value as input but does use the parameter to return a value. For an OUT parameter, set the in/out parameter type to "OUT".

See also

`getParamIOType()`

setParamName()

Sets the name of this stored-procedure parameter.

Syntax

```
void setParamName(String paramName)
```

Parameters

paramName The name of the stored-procedure parameter

Notes

The name of the parameter is informational only. It is used only for error messages and debugging. The parameter name is not needed to access the stored-procedure parameter because stored procedures are accessed by their index position in the stored-procedure declaration.

See also

`getParamName()`

setParamValue()

Sets the value of this stored-procedure parameter.

Syntax

```
void setParamValue(Object paramValue)
```

Parameters

paramValue The value of the stored-procedure parameter. The value must be a Java `Object` (such as `Integer`, `Double`, or `String`).

Notes

You must set the parameter value as a Java `Object`.

See also

`getParamValue()`

Part 4. Appendixes

© Copyright IBM Corp. 2003

Appendix A. Message files

Each map can have an associated message file. The *message file* contains the text for the map's exception and logging messages. A unique number identifies each message in the message file. The text of the message may also include placeholder variables, called *parameters*.

The methods that generate map messages provide two ways of generating the message text that a user sees. The coding of the method call can:
- Include the text of the message.
- Contain a reference to message text that is contained in an external message file.

It is generally a better practice for a map to refer to a message file than to generate the text itself, for ease of maintenance, administration, and internationalization.

This chapter describes message files, how they work, and how to set them up. It covers the following topics:

"Message location"	403
"Format for map messages" on page 405	405
"Message parameters" on page 406	406
"Maintaining the files" on page 407	407
"Operations that use message files" on page 407	407

Message location

All message file are located in the following directory of the IBM WebSphere InterChange Server Express product directory:

```
DLMs\messages
```

Note: In this document backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All IBM WebSphere InterChange Server Express product path names are relative to the directory where the IBM WebSphere InterChange Server Express product is installed on your system.

There are three types of message files that can be used to generate messages for a map:
- A map-specific message file, `mapName_locale.txt` where *mapName* corresponds to the name of the map and *locale* corresponds to the locale that the map is defined in.

  Map messages appear in the Messages tab of Map Designer and are stored as part of the map definition in the repository. When you compile the map, Map Designer extracts the message content and creates (or updates) the message file for runtime use. The name of the message file has the following format:

  ```
  MapName_locale.txt
  ```

  For example, for the `LegacyAddress_to_CwAddress` map, if it is created in an English locale in the United States, Map Designer creates the message file called `LegacyAddress_to_CwAddress_en_US.txt` and places it in the

ProjectName\Maps\Messages directory. After the map is deployed to InterChange Server Express, it will be placed in the DLMs\messages directory.

- The UserMapMessages.txt message file

  To this file, you can add new message numbers that fall into a "safe" range, as defined by IBM WebSphere InterChange Server Express (see Table 112). For example, if you create a message for an Oracle map, you would assign the message a number between 6101 and 6200. You can also use a message number that is already defined in the IBM WebSphere InterChange Server Express generic message file (CWMapMessages.txt, described next) and change the existing message text to text of your choice. Since the UserMapMessaages.txt file is searched *before* the IBM WebSphere InterChange Server Express message file, your additions override those messages.

- The IBM WebSphere InterChange Server Express generic message file, CWMapMessages.txt (which IBM WebSphere InterChange Server Express provides).

  If your map does *not* reference one of the other two message files, it must reference this one. Table 112 lists the message numbers that IBM WebSphere InterChange Server Express has assigned and that are contained in the generic message file.

  Attention: Do *not* change the contents of the IBM WebSphere InterChange Server Express generic message file CwMapMessages.txt! Make changes to a generic message by copying it into the UserMapMessage.txt message file and customizing it.

These files range from map-specific to general purpose. Messages that can be used by any map are located in a generic file, provided by IBM WebSphere InterChange Server Express. The other two files provide you with the option to customize messages for your maps, as needed.

Important: InterChange Server Express reads the UserMapMessages.txt and CWMapMessages.txt files into memory when it starts up. If you make changes to UserMapMessages.txt, you *must* restart InterChange Server Express for these changes to be available to maps.

Table 112. CwMapMessages.txt messages

Message number	Message text	Message usage
5000	Mapping - Value of the primary key in the source object is null. Map execution stopped.	Used if the primary key of the source object is null. The check for the source primary key = null should be always performed before any of the relationship methods are called that are based on the source object's primary key. If the key is null, the error should display and the map should stop execution.
5001	Mapping - RelationshipRuntimeException. Map execution stopped.	Used if RelationshipRuntimeException is caught in one of the following: • Function blocks – General/APIs/Identity Relationship/Maintain Simple Identity Relationship – General/APIs/Identity Relationship/Maintain Composite Relationship • Mapping APIs – maintainSimpleIdentityRelationship() – maintainCompositeRelationship()

Table 112. CwMapMessages.txt messages (continued)

Message number	Message text	Message usage
5002	Mapping - CxMissingIDException. Map execution stopped.	Used if `CxMissingIDException` is caught in one of the following: • Function blocks – General/APIs/Identity Relationship/Maintain Simple Identity Relationship – General/APIs/Identity Relationship/Maintain Composite Realtionship • Mapping APIs – `maintainSimpleIdentityRelationship()` – `maintainCompositeRelationship()`
5003	Mapping - Data in the {1} attribute is missing.	Used when the source attribute is null before using the function block Foreign Key Lookup (`foreignKeyLookup()`) or Foreign Key Cross-Reference (`foreignKeyXref()`). The check for the source attribute = null should be always performed before these relationship methods are called. If the key is null, the error should be displayed and the map might stop execution.
5007	Mapping - ForeignKeyLookup() of '{1}' with Source Value of '{2}' failed for the '{3}' relationship and '{4}' participant on Initiator '{5}'. Map execution stopped.	Used if the destination attribute is null after using the function block Foreign Key Lookup (`foreignKeyLookup()`). Map has to stop execution.
5008	Mapping - ForeignKeyLookup() of '{1}' with Source Value of '{2}' failed for the '{3}' relationship and '{4}' participant on Initiator '{5}'. Map execution continued.	Used if the destination attribute is null after using the function block Foreign Key Lookup (`foreignKeyLookup()`). Map has to continue execution.
5009	Mapping - ForeignKeyXref() of '{1}' with Source Value of '{2}' failed for the '{3}' relationship and '{4}' participant on Initiator '{5}'. Map execution stopped.	Used if the destination attribute is null after using the function block Foreign Key Cross-Reference (`foreignKeyXref()`). Map has to stop execution.

When a map references a message number, the message files are searched in the following order:

1. The map-specific message file *mapName_locale*.txt where *mapName* corresponds to the name of the map, is searched.
2. The file `UserMapMessages.txt` is searched.
3. The IBM WebSphere InterChange Server Express generic message `CWMapMessages.txt` is searched.

Format for map messages

To ensure consistency of messages, IBM WebSphere InterChange Server Express has developed a message format. This section describes that format, including:

- "Message format" on page 406
- "Message parameters" on page 406
- "Comments" on page 407

Note: The map-specific message file should be modified from the message tab in Map Designer Express and should not be modified directly. Map Designer Express will overwrite any custom modification in the map-specific message

file with the messages saved in the map. However, for the message files `UserMapMessages.txt` and `CWMapMessages.txt`, it is safe to modify the file directly.

Message format

The format for each message is:

```
MessageNum
Message
```

The message number (*MessageNum*) and the message itself (*Message*) must be on different lines, with a carriage return at the end of each line.

For example, a map's messages might include a message identified as number 23, whose text includes two placeholder variables, marked as {1} and {2}, as shown in Figure 119.

```
23
Customer ID {1} could not be changed: {2}
```

Figure 119. Sample Message

Message parameters

When the map calls a method that displays a particular message, it passes to the method the message's identifying number and potentially additional parameters. The method uses the identifying number to locate the correct message in the message file, and it inserts the values of the additional parameters into the message text's placeholder variables.

It is not necessary to write separate messages for each possible situation. Instead, use parameters to represent values that change at runtime. The use of parameters allows each message to serve multiple situations and helps to keep the message file small.

A parameter always appears as a number surrounded by curly braces: {*number*}. For each parameter you want to add to the message, insert the number within curly braces into the text of the message, as follows:

```
message text {number} more message text.
```

For example, consider message 23 in Figure 119 again. When the map wants to display or log this message, it passes to the appropriate method the identifying number of the message (23) and two additional parameters:

- Parameter 1 becomes the customer ID number (6701)
- Parameter 2 becomes a `String` variable containing some additional explanatory text, such as `greater than maximum length`.

The method locates the correct message, substitutes the parameter values for the message's placeholders, and displays or logs the following message:

```
Customer ID 6701 could not be changed: greater
than maximum length
```

Because the message text takes the description of the missing entry and its ID as parameters, rather than including them as hardcoded strings, you can use the same message for any pair of customer ID and explanatory text.

Comments

Precede each comment line in a message file with a pound sign (#). For example, a comment might look like this:

```
# Message file for the Address business object map.
```

It is good practice to start the file with a series of comment lines to form a short header. Include in the header data the name of the map and such information as the file creator and file creation date.

Maintaining the files

At a user site, an administrator might set up a procedure for filtering map messages and notifying someone who can resolve problems, by e-mail or e-mail pager. This means that the error numbers and the meanings associated with the numbers must remain the same after the first release of a map.

You can change the text associated with an error number, but you should avoid changing the meaning of the text or reassigning error numbers. If you do change the meanings associated with error numbers, you should document the change and notify users of the map.

Operations that use message files

Message files hold text for messages used in several types of operations. Table 8 on page 21 lists the types of operations that use message files and the methods of the `BaseDLM` class that perform those operations.

Table 113. Message-generating operations

Operation	Function block	Method
Raising exceptions	General/APIs/Maps/Exception/ Raise Map Exception	`raiseException()`
Logging	• General/Logging and Tracing/Log Information ID • General/Logging and Tracing/Log error ID • General/Logging and Tracing/Log warning ID	• `logInfo()` • `logError()` • `logWarning()`
Tracing	General/Logging and Tracing/Trace/Trace on Level	`trace()`

This section describes message-generating operations that affect map execution.

Raising exceptions

The `raiseException()` method has several forms. One commonly used syntax is:

```
raiseException(String exceptionType,
   int messageNum, String param[,...])
```

With this syntax, you can have from one to three *param* `String` parameters. Thus, there can be up to five comma-separated parameters in a call to `raiseException()`.

This example raises a new exception, using message number 23, and passes in two parameters to the message, the customer ID value and a string:

```
raiseException(AttributeException, 23,
   fromCustomer.getString("CustomerID"),
   "greater than maximum length");
```

Figure 119 shows the text for message 23 as it appears in the message file.

Logging messages

A map can log a message whenever something occurs that might be of interest to an administrator. To log a message, a map uses the `logInfo()`, `logWarning()`, and `logError()` methods of the `BaseDLM` class. Each method is associated with a different message severity level.

Severity levels

To log a message, you must call the method associated with the message's severity level. Table 114 lists the severity levels and their associated methods.

Table 114. Message levels

Severity level	Method	Description
Info	`logInfo()`	Informational only. The user does not need to take action.
Warning	`logWarning()`	Represents information about a problem. Do not use this level for problems that the user must resolve.
Error	`logError()`	Indicates a serious problem that the user needs to investigate.

Using a message file

Every map has at least one message file associated with it. If a map does not use custom messages, its messages come from the system map message file, `CWMapMessages.txt`. If a map uses customized messages, it has a map-specific message file (which is generated from the messages entered in the Messages tab of Map Designer). For more information, see "Message location" on page 403.

When a map logs an error, the text of the error message comes from the map's message file. The following example logs an error message whose text is contained in the map's message file. The text of error message 10 appears as follows in the message file:

```
10
Credit report error for {1}, {2}.
```

The code to log the message looks like this:

```
logError(10, customer.get("LName"), customer.get("FName"));
```

When the `logError()` method executes, the text for message 10 is written to the log file, with the customer's last name and first name substituted for parameters 1 and 2. For example, the logged message for a customer named John Davidson looks like this:

```
Credit report error for Davidson, John.
```

Principles of good message logging

When creating messages, be sensitive to the way that administrators use the logging feature.

Assigning severity levels: It is important to be precise when assigning error levels to messages. The IBM system e-mail notification feature sends a message to a designated person, usually the administrator, when it detects the generation of an

error message or fatal error message. Administrators use this IBM system e-mail notification feature, and they additionally might link it to an e-mail pager to send a page when an error occurs. By being precise when assigning error levels to messages, you can reduce the number of critical messages.

Revising messages: You can revise the text of a message at any time, such as to clarify or expand the text. However, once you assign a message number to a certain type of error, it is important that you do not reassign the number. Many administrators depend on scripts to filter log messages, and these scripts rely on the message numbers. Thus, it is important that the numbers in the message file do not change meaning. If they do, users can lose messages or receive inadvertent messages.

When to use informational messages: You can use the `logInfo()` method to create temporary messages for your own debugging. However, be sure to remove these debugging method calls when you are finished with development.

Resist the temptation to use the `logInfo()` method to document the normal operation of the collaboration. Doing so fills the administrator's log files with messages that are not of interest. Instead, use the `trace()` method to give the administrator detailed information for debugging.

Adding trace messages

You can add trace messages to your map so that when a map instance runs, it generates a detailed description of its actions. Trace messages are useful for your own debugging and for on-site troubleshooting by administrators.

Trace messages differ from log messages in that trace messages are suppressed by default, whereas log messages cannot be suppressed. Trace messages are generally more detailed and are meant to be viewed only under certain circumstances, such as when someone intentionally configures the map's trace level to a number higher than zero. You can send trace messages and log messages to different files.

You can add trace messages to a map to report operations that are specific to that map. These are some types of information that the map can write to the trace file:
- Key values of a business object at the point that the map begins or ends a particular transformation step.
- The decision to take a particular branch in the execution path.

Assigning trace levels

Each trace message must be associated with a **trace level** between 1 and 5. The trace level usually correlates to a level of detail: messages at level 1 typically contain less detail than messages at level 2, which contain less detail than those at level 3, and so forth. Thus, if you turn on tracing at level 1, you see messages that contain less detail than the messages at level 5. However, you can assign levels in any way that is useful to you. Here are some suggestions:
- You can assign the same level to all of your trace messages.
- You can assign trace levels according to level of detail.
- You can assign message levels according to the business object involved: level 1 traces messages relating to a certain business object, level 2 traces messages relating to another business object, and so on.

When you turn on tracing at a particular level, the messages associated with the specified level and those associated with all lower levels appear. For example, tracing at level 2 displays messages associated with both level 2 and level 1.

Tip: Make sure to note the tracing levels with your documentation, so users know what level to use when they need to trace.

Generating a trace message

The following is an example of a message and the method call that generates the message. The message appears in the message file as follows:

```
20
Begin transformation on {1} attribute: value = {2}
```

The method call obtains the value of the attribute `LName`, then uses the value to replace the parameter in the message. The code appears in the map as follows, and the message appears when the user sets tracing to level 3:

```
trace(3, 20, "LName", customer.get("LName"));
```

Setting the trace level

Figure 120 shows the General tab of the Map Properties dialog in Map Designer. (For information on how to display the Map Properties dialog, see "Providing map property information" on page 54.) Notice that you can set the trace level for trace messages in this dialog.

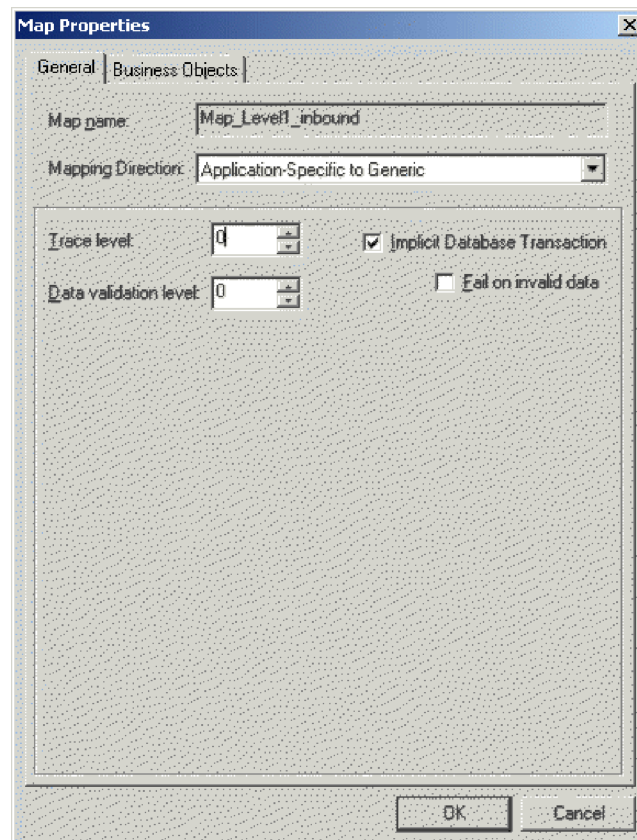


Figure 120. Trace level for a map

As the map developer, you create the levels for which map-generated tracing can be requested, as described in "Assigning trace levels" on page 409.

Note: If you change the trace level for an activated map, you must stop and restart the map before the new trace level takes effect. Use the Component menu of IBM WebSphere System Manager to stop and start a map.

By setting the trace level in the Map Properties dialog of Map Designer, you set it for *all* map instances based on this map definition. You can also set the trace level for all map instances from the Map Properties window of IBM WebSphere System Manager. For more information about the Map Properties window of System Manager, see the *User Guide*.

Appendix B. Attribute properties

Table 115 lists the properties for attributes of business object definitions.

Table 115. Attribute Properties

Property	Description
Name	A name that describes what type of data the attribute contains. The name can be up to 80 alphanumeric characters and underscores. It cannot contain spaces or other punctuation.
Type	The data type of the attribute. Basic types include String, Boolean, Double, Float, Integer, and Date. If the attribute references a child business object, specify the name of a child business object definition. Attributes that reference child business objects are called compound attributes.
IsKey	A boolean value, `true` or `false`, specifying whether this is a key attribute. Key attributes uniquely identify a business object created from the definition. Each business object definition has at least one key attribute.
IsForeignKey	A boolean value, `true` or `false`, specifying whether this is a foreign key attribute.
MaxLength	An integer representing the maximum number of bytes the attribute can contain. To specify no limit, enter zero (`0`).
AppSpecificInfo	A string that provides information about the attribute for a particular application, such as the name of a field in a table or form that corresponds to the attribute. Connectors use this information when processing the object.
DefaultValue	The value to assign to this attribute if there is no runtime value.
IsRequired	A boolean value, `true` or `false`, specifying whether a value for this attribute is required to create a business object.
ContainedObjectVersion	The version number of the child business object definition. IBM WebSphere System Manager displays this value under the name Type Version.
Relationship	The relationship between the parent business object and the child business object. In the current release, the only valid relationship is Containment.
Cardinality	The number of child business objects that this attribute references. If the attribute references only one child business object, the value is 1. If the attribute can reference many child business objects, the value is a literal `n`.

© Copyright IBM Corp. 2003

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

© Copyright IBM Corp. 2003

Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others. Map Designer Express and Relationship Designer Express include software developed by the Eclipse Project (http://www.eclipse.org/).

WebSphere Business Integration t Express for Item Synchronization V4.3

Index

Special characters

.bo file extension 12, 74, 75, 79
.class file extension 12, 70
.cwm file extension 12, 48, 53
.java file extension 12, 70, 71
.txt file extension 12, 67, 221, 403

A

Access client 72, 148, 193
ACCESS_REQUEST calling context 147, 148, 193
  Create verb and 193, 214, 218
  Delete verb and 194, 214
  foreignKeyXref() and 217, 351
  getOriginalRequestBO() and 365
  maintainChildVerb() and 214, 354
  maintainCompositeRelationship() and 204, 355
  maintainSimpleIdentityRelationship() and 193, 358
  original-request business object 148, 365
  Retrieve verb and 194, 214, 218
  retrieving 364
  setting to 367
  testing with 81, 84
  Update verb and 194, 214, 218
ACCESS_RESPONSE calling context 147, 148, 193
  foreignKeyXref() and 220, 351
  getOriginalRequestBO() and 366
  maintainCompositeRelationship() and 204, 355
  maintainSimpleIdentityRelationship() and 200, 358
  original-request business object 148, 201, 366
  retrieving 364
  setting to 367
  updateMyChildren() and 210
Activity Editor 87
  accessing 25, 26, 37, 40, 41, 45, 88
  Add Comment 90
  Add Description 90, 131
  Add Label 90
  Add To do 90
  Add To My Collection 90
  Comment 94
  connection links 93
  Content window 92
  Cross-Reference transformation 88
  Description 94
  Design mode 92, 135
  Document Display Area 88
  example of using 122, 126, 133
  function blocks 93, 95
  Graphical view 88, 91
  grouping components 94
  Help menu 90
  Java view 88, 135
  Join transformation 40, 88
  keyboard shortcuts 88
  Label 94
  layout 88
  Library window 92
  main menus 88
  main views 88
Activity Editor *(continued)*
  New Constant 90, 94
  ports 93
  Properties window 92
  Quick view mode 92, 136
  Resize label 94
  Set Value transformation 37, 88
  Split transformation 41, 88
  starting 87
  Status bar 91
  Submap transformation 45, 88
  Title Bar 88
  To Do 94
  toolbars 90
addDays() method 310
addElement() method 260
addMyChildren() method 345, 370, 391
addParticipant() method 378
addWeekdays() method 311
addYears() method 312
after() method 313
AnyException exception 234
Application-specific business objects 3
AppSpecificInfo attribute property 413
Attribute
  addressing in transformations 140
  advanced settings 182
  application-specific information 413
  checking for key 248
  column name 182
  comments for 17, 34, 50, 57, 63
  data type 16, 34, 256, 413
  dependencies of 66
  destination 5, 16
  finding 36, 49, 60
  joining 38
  maximum length 413
  name 16, 34, 413
  properties 413, 415
  relationship 146, 187, 188
  required 250, 413
  source 16
  specifying 240
  splitting 40
  unlinked 18, 49, 59, 60
  validating 52, 69
Attribute value
  adding together 270
  blank 248
  copying 37, 50
  default 36, 73, 253, 413
  null 249
  retrieving 245
  retrieving as string 255
  retrieving maximum 262, 263, 264
  retrieving minimum 265, 266, 267
  setting 251, 254
  setting default value for 253
  validating 146
  validating data type 256
  zero-length string 248

© Copyright IBM Corp. 2003

AttributeException exception 234

B

BaseDLM class 227, 237
  defined 227
  getDBConnection() 227
  getName() 229
  getRelConnection() 230
  implicitDBTransactionBracketing() 231
  isTraceEnabled() 231
  logError() 232
  logInfo() 232
  logWarning() 232
  method summary 227
  releaseRelConnection() 235
  trace() 236
before() method 314
beginTran() method (deprecated) 289
beginTransaction() method 273
Blank attribute value 248
BOOL_TYPE constant 298
Boolean class 413
  as stored-procedure parameter type 285
  converting to 300
  converting to Boolean 303
  determining data type 298
  valid conversions 300
boolean data type
  as stored-procedure parameter type 285
  checking for valid data 256
  converting to 303
  converting to Boolean 300
  determining data type 298
  getting attribute value 245
  setting attribute to 251
  valid conversions 300
Breakpoints 75, 78
Browsing a Project 169
Business object
  adding 14, 169
  adding to an array 260
  addressing in transformations 140
  business object definition for 247
  comparing attribute values 243, 244
  comparing key attribute values 242
  copying 241
  deleting 14, 63, 169
  duplicating 242
  generic 3, 148
  instance name 34, 140
  key attribute in 248
  null attribute in 249
  number in a business object array 270
  properties 141
  refreshing list of 17
  removing from business object array 268, 269
  required attribute in 250
  retrieving attribute value 245, 255
  retrieving from business object array 261
  retrieving key attribute value 250
  retrieving verb 247
  setting attribute value 251, 252, 254
  setting key values 253
  setting value of 269
  swapping in an array 270
  temporary 142
Business object *(continued)*
  transversing hierarchical 240
  validating attribute data type 256
  variable for 140
Business object array
  adding attribute values together 270
  adding business object to 260
  comparing with another 261
  duplicating 260
  index 69, 74, 140
  removing all elements from 268
  removing element from 268, 269
  retrieving a business object from 261
  retrieving contents of 262
  retrieving last index of 262
  retrieving maximum attribute value from 262, 263, 264
  retrieving minimum attribute value from 265, 266, 267
  retrieving size of 270
  retrieving values as string 271
  reversing position of elements in 270
  setting element of 269
Business Object Array function block 96
Business object definition
  retrieving name of 247
Business Object function block 98
Business Object/Array function block 97
Business Object/Constants function block 97
BusObj class 239, 257
  copy() 241
  defined 239
  deprecated methods 256
  duplicate() 242
  equalKeys() 242
  equals() 243
  equalsShallow() 244
  exists() 244
  getCount() 257
  getKeys() 257
  getLocale() 247, 253
  getType() 247
  getValues() 257
  getVerb() 247
  isBlank() 248
  isKey() 248
  isNull() 249
  isRequired() 250
  keysToString() 250
  method summary 239
  not() 257
  set() 251, 257
  setContent() 252
  setDefaultAttrValues() 253
  setKeys() 253
  setVerb() 254
  setWithCreate() 254
  toString() 255
  validData() 256
BusObjArray class 259, 271
  addElement() 260
  defined 259
  duplicate() 260
  elementAt() 261
  equals() 261
  getElements() 262
  getLastIndex() 262
  max() 262
  maxBusObjArray() 263

BusObjArray class *(continued)*
maxBusObjs() 264
method summary 259
min() 265
minBusObjArray() 266
minBusObjs() 267
removeAllElements() 268
removeElement() 268
removeElementAt() 269
setElementAt() 269
size() 270
sum() 270
swap() 270
toString() 271

C

calcDays() method 314
calcWeekdays() method 315
CALL statement 276, 277, 291
Call-triggered flow 148
Calling contexts 146
ACCESS_REQUEST 147, 193
ACCESS_RESPONSE 147, 193
EVENT_DELIVERY 147, 193
example of 149
identity relationship and 193
retrieving 363
SERVICE_CALL_FAILURE 147, 193
SERVICE_CALL_REQUEST 147, 193
SERVICE_CALL_RESPONSE 147, 193
setting 366
testing with 80
CANNOTCONVERT constant 299
Cardinality attribute property 413
child business objects
customizing for relationships 205
example of customizing for relationships 205
Child business objects
adding to parent/child relationship 345, 358
attribute comment for 50
cardinality of 176, 413
identity relationships 176
multiple-cardinality 42
removing from parent/child relationship 347, 358
setting verb for 352
submaps for 42, 44
testing 74
verb 213
version number 413
CLASSPATH environment variable 140
CollaborationException class 240
commit() method (CwDBConnection) 274
commit() method (DtpConnection) 235, 290
Comparing
business object arrays 261
business object attribute values 243, 244
key attribute values 242
Composite identity relationship 157, 159, 174, 202, 210
customizing map rules for 204
defining 175, 176, 202
main map 205
maintainChildVerb() and 206, 216
maintainCompositeRelationship() and 203, 354
managing child instances 207
participant type for 202
Connection
determining if active 280
obtaining 227
releasing 281
transaction programming model 227, 228
Connection pool 228, 281
Connector
initiating mapping request 72, 147, 193
retrieving name of 363
setting name of 366
ContainedObjectVersion attribute property 413
Context menu (Activity Editor) 90
Add Comment 90
Add Description 90
Add Label 90
Add To do 90
Add To My Collection 90
New Constant 90
Context menu (business object browser)
Copy 34
Refresh All 17
Context menu (business object pane)
Add Business Object 33
Delete Business Object 63
Context menu (business object window)
Delete 35
Properties 142
Context menu (dest. data, attribute)
Clear Breakpoint 77
Set Breakpoint 76
Context menu (dest. data, main object)
Collapse 76
Save To 79
Context menu (map workspace)
Add Business Object 33
Delete 63
Map Properties 54
Paste As Input Object 34
Paste As Output Object 34
Context menu (Relationship Designer)
Change Index 176
Context menu (source data, child object)
Add Instance 73, 74, 75
Remove All Instances 75
Remove Instance 75
Context menu (source data, main object)
Load From 75
Reset 73
Save To 74
Context menu (Transformations)
Open 25
Open in New Window 25
View Source 26
copy() method 241, 257
Copying
attributes 37, 50
business object 241
participant definitions 178, 179
relationship definitions 178
Create verb
conditionally set 211
foreignKeyXref() and 218, 220
maintainChildVerb() and 214, 215
maintainCompositeRelationship() and 204
maintainSimpleIdentityRelationship() and 193, 195, 197, 200
create() method 370, 374, 380

Cross-Reference transformation 16, 21, 36, 45, 69, 88
defining for relationships 191
validating 52
Custom transformation 16, 21, 36, 46, 50, 88, 187
CwDBConnection class 273, 283
beginTransaction() 273
commit() 274
creating object of 227
executePreparedSQL() 275
executeSQL() 276
executeStoredProcedure() 278
getUpdateCount() 279
hasMoreRows() 279
inTransaction() 280
isActive() 280
method summary 273
nextRow() 281
release() 281
rollBack() 282
CwDBStoredProcedureParam class 285, 287
constructor 285
getParamType() 286
getValue() 287
method summary 285
CwDBStoredProcedureParam() constructor 285
CwDBTransactionException exception 228, 274, 275, 282, 283
cwExecCtx variable 146, 333, 346, 347, 349, 350, 355, 357, 359
CWMapMessages.txt message file 404
CWMAPTYPE constant 333
CxMissingIDException exception 405

D

Data conversion 38, 297
class for 297
Java.lang methods 297
to boolean data type 303
to Boolean object 300
to double data type 303
to Double object 301
to float data type 304
to Float object 301
to int data type 304
to Integer object 302
to String object 305
valid conversions 300
Data type
attribute 413
determining 297
determining if conversion is possible 298
Data validation 146
Database
connecting to 227
executing a query in 276, 277, 278
querying 279, 281
rows affected by last write 279
Database Connection function block 101
DataValidationLevel map property 85
Date class 285, 298, 300, 413
Date formatting
adding days to date 310
adding weekdays to date 311
adding years to date 312
calculating days between dates 314
calculating weekdays between dates 315
comparing dates 313, 314
current date 309
Date formatting *(continued)*
generic format 317
getting day of the month 317, 318
getting day of the week 318, 319
getting earliest date from a list 323, 325
getting hour value 318
getting in specified or default format 330
getting milliseconds between 1/1/70 and date 321
getting minutes value 319, 326
getting month name 326, 327
getting month value 320, 326
getting most recent date from a list 321, 322
getting seconds value 320, 327
getting year 320, 328
parsing date according to format 309
reformatting to CrossWorlds date format 317
using full names of months 316, 328, 329
using short names of months 316, 329
using weekday names 316, 330
Date function block 111, 127
DATE_TYPE constant 298
Date/Formats function block 113
deactivateParticipant() method 381
deactivateParticipantByInstance() method 382
Debug menu (Map Designer) 25
Advanced 25
Attach 25, 79
Breakpoints 25, 77
Clear All Breakpoints 25, 77
Continue 25, 78
Detach 25, 79
Run Test 25, 78
Step Over 25, 78
Stop Test Run 25
Toggle Breakpoint 25, 76
Default attribute value 36, 253, 413
DefaultValue attribute property 413
DELETE statement 276, 277
Delete verb
foreignKeyXref() and 220
maintainChildVerb() and 214, 215
maintainCompositeRelationship() and 204
maintainSimpleIdentityRelationship() and 194, 196, 197, 200
deleteMyChildren() method 347
deleteParticipant() method 383
deleteParticipantByInstance() method 384
Deprecated methods
BusObj class 256, 368
DtpConnection class 289
Relationship class 390
UserStoredProcedureParam class 393
Design mode (Activity Editor) 92
Designer toolbar (Map Designer) 22
Add Business Object 33
All Attributes 22
Clear All Breakpoints 77
Compile 70
Continue 78
displaying 22, 24
Linked Attributes 22
Run Test 78
Step Over 78
Toggle Breakpoint 76
Unlinked Attributes 22
Validate 69
Destination business object 3, 5, 13, 144

Destination business object *(continued)*
adding to map 30, 33
business object window 34
displaying 9, 17, 24, 55
execution order 16, 52, 66, 69
relationship and 153
setting verb of 35
variable for 140
verb 35, 210
Diagram tab (Map Designer) 17
adding business object 33
business object browser 17, 22, 24
business object variables 141
business object window 18, 24, 34, 141
calling a submap 44
custom transformation 46
default display 22
deleting a transformation 63
displaying attributes 36
joining attributes 38
key mappings 21
map workspace 18, 143
moving attribute 37
setting attribute value 36
splitting attribute 40
temporary business object 143
Double class 413
as stored-procedure parameter type 285
converting to 301
converting to Double 303
converting to Float 302, 304
converting to Integer 302, 305
converting to String 305
determining data type 298
obtaining maximum value 263, 264
obtaining minimum value 265, 266, 267
valid conversions 300
double data type
as stored-procedure parameter type 285
checking for valid data 256
converting to 303
converting to Double 301
converting to Float 302, 304
converting to Integer 302, 305
converting to String 305
determining data type 298
getting attribute value 245
setting attribute to 251
valid conversions 300
DOUBLE_TYPE constant 298
DtpConnection class (deprecated) 289, 295
beginTran() 289
commit() 290
creating object of 230
execStoredProcedure() 292
executeSQL() 291
getUpdateCount() 293
hasMoreRows() 293
inTransaction() 294
method summary 289
nextRow() 294
rollBack() 295
DtpDataConversion class 297, 306
CANNOTCONVERT 299
defined 297
getType() 297
isOKToConvert() 298
DtpDataConversion class *(continued)*
method summary 297
OKTOCONVERT 299
POTENTIALDATALOSS 299
toBoolean() 300
toDouble() 301
toFloat() 301
toInteger() 302
toPrimitiveBoolean() 303
toPrimitiveDouble() 303
toPrimitiveFloat() 304
toPrimitiveInt() 304
toString() 305
DtpDate class 307, 331
addDays() 310
addWeekdays() 311
addYears() 312
after() 313
before() 314
calcDays() 314
calcWeekdays() 315
DtpDate() 309
get12MonthNames() 316
get12shortMonthNames() 316
get7DayNames() 316
getCWDate() 317
getDayOfMonth() 317
getDayOfWeek() 318
getHours() 318
getIntDay() 318
getIntDayOfWeek() 319
getIntMilliSeconds() 319
getIntMinutes() 319
getIntMonth() 320
getIntSeconds() 320
getIntYear() 320
getMaxDate() 321
getMaxDateBO() 322
getMinDate() 323
getMinDateBO() 325
getMinutes() 326
getMonth() 326
getMSSince1970() 321
getNumericMonth() 326
getSeconds() 327
getShortMonth() 327
getYear() 328
method summary 307
rules for 307
set12MonthNames() 328
set12MonthNamesToDefault() 329
set12ShortMonthNames() 329
set12ShortMonthNamesToDefault() 329
set7DayNames() 330
set7DayNamesToDefault() 330
toString() 330
DtpDate() constructor 309
DtpMapService class 333, 334
method summary 333
runMap() 333
DtpSplitString class 335, 340
defined 335
DtpSplitString() 335
elementAt() 336
firstElement() 336
getElementCount() 337
getEnumeration() 338

DtpSplitString class *(continued)*
lastElement() 338
method summary 335
nextElement() 338
prevElement() 339
reset() 340
DtpSplitString() constructor 335
DtpUtils class 341, 343
method summary 341
padLeft() 341
padRight() 341
stringReplace() 342
truncate() 343
duplicate() method 242, 260
Duplicating
business object 242
business object array 260
Dynamic relationship 184

E

Edit menu (Activity Editor) 89
Copy 89
Cut 89
Delete 89
Find 89
Goto Line 89
Paste 89
Select All 89
Edit menu (Map Designer) 24
Add Business Object 24, 32, 141, 142
Delete Business Object 24, 63
Delete Current Selection 24, 34, 63
Find 24, 36, 49, 60
Insert Row 24
Map Properties 24, 54, 85, 141
Replace 24, 62
Select All 24
Edit menu (Relationship Designer) 172
Advanced Settings 172, 175, 177, 179, 182
Copy 172, 178, 179
Cut 172
Delete 183
Paste 172, 178, 179
Rename 172
elementAt() method 261, 336
Environment variable
CLASSPATH 140
JCLASSES 139
PATH 10
equalKeys() method 242
equals() method 243, 261
equalsShallow() method 244
Error
compilation 71
run-time 85
Error message 232, 408
EVENT_DELIVERY calling context 147, 193
Create verb and 193, 214, 218
Delete verb and 194, 214
foreignKeyXref() and 217, 351
getOriginalRequestBO() and 365
maintainChildVerb() and 214, 354
maintainCompositeRelationship() and 204, 355
maintainSimpleIdentityRelationship() and 193, 358
original-request business object 148, 365
Retrieve verb and 194, 214, 218
EVENT_DELIVERY calling context *(continued)*
retrieving 364
setting to 367
testing with 81, 84
Update verb and 194, 214, 218
updateMyChildren() and 210
Event-triggered flow 147
Exception handling 144
Exception types 240
Exceptions
CollaborationException class 240
CwDBTransactionException 228, 274, 275, 282, 283
defined 144, 240
raising 233, 407
RelationshipRuntimeException class 145
relationships 145
type 240
execStoredProcedure() method (deprecated) 292
executePreparedSQL() method 275
executeSQL() method (CwDBConnection) 276
executeSQL() method (DtpConnection) 291
executeStoredProcedure() method 278
exists() method 244

F

File menu (Activity Editor) 89
Close 89
Print 89
Print Preview 89
Print Setup 89
Save 89
File menu (Map Designer) 23
Close 23, 53
Compile 23, 53, 70, 73
Compile All 24, 70
Compile with Submap(s) 24, 70
Create Map Document 24, 58
Delete 23, 64
Exit 24, 53
New 23, 29
Open 23, 52, 53
Print 24, 62
Print Preview 24, 62
Print Setup 24, 62
Save 23, 47, 49
Save As 23, 47, 49
Validate Map 23, 69
View Map Document 24, 60
File menu (Relationship Designer) 171
Add Participant Definition 172, 173
New 171
New Relationship Definition 173
Save 171
Save All 171
Save Relationship Definition 174, 178, 179
Switch to Project 171
Find and Replace text 62
Find text 60
Find unlinked attribute 60
firstElement() method 336
Float class 413
as stored-procedure parameter type 285
converting to 301
converting to Double 301, 303
converting to Float 304
converting to Integer 302, 305

Float class *(continued)*
converting to String 305
determining data type 298
obtaining maximum value 263, 264
obtaining minimum value 265, 266, 267
valid conversions 300
float data type
as stored-procedure parameter type 285
checking for valid data 256
converting to 304
converting to Double 301, 303
converting to Float 302
converting to Integer 302, 305
converting to String 305
determining data type 298
getting attribute value 245
setting attribute to 251
valid conversions 300
FLOAT_TYPE constant 298
Foreign key 216, 348, 350, 413
Foreign Key Cross-Reference function block 217
Foreign key lookup 216
Foreign Key Lookup function block 216
foreignKeyLookup() method 216, 348, 405
foreignKeyXref() method 217, 350, 405
Function blocks 93, 95
adding custom Jar libraries as 137
customizing Jar library properites 137
example of using 122, 123, 127, 133
General/APIs/Business Object 98
General/APIs/Business Object Array 96
General/APIs/Business Object/Array 97
General/APIs/Business Object/Constants 97
General/APIs/Database Connection 101
General/APIs/Identity Relationship 103
General/APIs/Maps 106
General/APIs/Maps/Constants 104
General/APIs/Maps/Exception 105
General/APIs/Participant 107
General/APIs/Participant/Array 106
General/APIs/Participant/Constants 107
General/APIs/Relationship 109
General/Date 111
General/Date/Formats 113
General/Logging and tracing 113
General/Logging and Tracing/Log Error 113
General/Logging and Tracing/Log Information 114
General/Logging and Tracing/Log Warning 114
General/Logging and Tracing/Trace 115
General/Mapping 116
General/Math 116
General/Properties 118
General/Relationship 118
General/String 119
General/Utilities 121
General/Utilities/Vector 121
using to implement relationships 187

G

get12MonthNames() method 316
get12ShortMonthNames() method 316
get7DayNames() method 316
getConnName() method 363
getCount() method (deprecated) 257
getCWDate() method 317
getDayOfMonth() method 317
getDayOfWeek() method 318
getDBConnection() method 227, 228
getElementCount() method 337
getElements() method 262
getEnumeration() method 338
getGenericBO() method (deprecated) 368
getHours() method 318
getInitiator() method 363
getInstanceId() method 371
getIntDay() method 318
getIntDayOfWeek() method 319
getIntMilliSeconds() method 319
getIntMinutes() method 319
getIntMonth() method 320
getIntSeconds() method 320
getIntYear() method 320
getKeys() method (deprecated) 257
getLastIndex() method 262
getLocale() method 247, 253, 364
getMaxDate() method 321
getMaxDateBO() method 322
getMinDate() method 323
getMinDateBO() method 325
getMinutes() method 326
getMonth() method 326
getMSSince1970() method 321
getName() method 229
getNewID() method 385
getNumericMonth() method 326
getOriginalRequestBO() method 365, 368
getParamDataTypeJavaObj() method (deprecated) 394
getParamDataTypeJDBC() method (deprecated) 395
getParamIndex() method (deprecated) 395
getParamIOType() method (deprecated) 396
getParamName() method (deprecated) 397
getParamType() method 286
getParamValue() method (deprecated) 397
getParticipantDefinition() method 372
getRelationshipDefinition() method 372
getRelConnection() method (deprecated) 230, 291, 292
getSeconds() method 327
getShortMonth() method 327
getType() method 247, 297
getUpdateCount() method (CwDBConnection) 279
getUpdateCount() method (DtpConnection) 293
getValue() method 287
getValues() method (deprecated) 257
getVerb() method 247
getYear() method 328
Graphical view (Activity Editor) 88, 91
Content window 92
Design mode 92
Library window 92
Properties window 92
Quick view mode 92
Graphics toolbar (Activity Editor) 91
Back 91
Forward 91
Home 91
Up One Level 91
Zoom In 91
Zoom Out 91

H

hasMoreRows() method (CwDBConnection) 279
hasMoreRows() method (DtpConnection) 293

Help menu (Activity Editor) 90
Help menu (Map Designer) 25
Help menu (Relationship Designer) 172
Hierarchical business object
  comparing all 243
  comparing top-level 244
  transversing 240

I

Identity relationship 155, 159
  adding child business objects 345, 358
  child business objects 176
  class for 345
  creating participant for 370
  defined 154, 155, 174
  defining 174, 176, 191, 201, 202
  deleting child business objects 347, 358
  kinds of 155, 174
  maintaining child verb 213
  relationship instance IDs 162
  static 185
  static lookup 211
  testing 80
Identity Relationship function block 103
IdentityRelationship class 161, 162, 345, 361
  addMyChildren() 345, 391
  deleteMyChildren() 347
  foreignKeyLookup() 348
  foreignKeyXref() 350
  maintainChildVerb() 352
  maintainCompositeRelationship() 354, 391
  maintainSimpleIdentityRelationship() 356, 391
  method summary 345
  updateMyChildren() 358, 391
implicitDBTransactionBracketing() method 231
IN parameter 287
Inbound map 3, 4
  example of customizing 211
  foreign key lookup in 219, 349, 351
  in map document 57
  lookup relationship in 387
  testing 81, 82, 84
Informational message 232, 408, 409
INOUT parameter 287
INSERT statement 190, 276, 277, 279
int data type
  as stored-procedure parameter type 285
  checking for valid data 256
  converting to 304
  converting to Double 301, 303
  converting to Float 302, 304
  converting to Integer 302
  converting to String 305
  determining data type 298
  getting attribute value 245
  setting attribute to 251
  valid conversions 300
Integer class 413
  as stored-procedure parameter type 285
  converting to 302
  converting to Double 301, 303
  converting to Float 302, 304
  converting to Integer 305
  converting to String 305
  determining data type 298
  obtaining maximum value 263, 264
Integer class *(continued)*
  obtaining minimum value 265, 266, 267
  valid conversions 300
INTEGER_TYPE constant 298
inTransaction() method (CwDBConnection) 280
inTransaction() method (DtpConnection) 294
INVALID_INSTANCE_ID constant 372, 374, 380
isActive() method 280
isBlank() method 248
IsForeignKey attribute property 413
IsKey attribute property 413
isKey() method 248
isNull() method 249
isOKToConvert() method 298
IsRequired attribute property 413
isRequired() method 250
isTraceEnabled() method 231

J

Jar libraries
  customizing display settings 137
  importing as function blocks 137
Java class
  Boolean 300, 413
  Date 298, 413
  Double 301, 303, 413
  Float 301, 304, 413
  Integer 302, 305, 413
  Object 245, 251, 256
  StringTokenizer 335
  Vector 276, 281, 286
Java compiler (javac) 70
Java Development Kit (JDK) 10
Java operator
  NOT 257
Java view (Activity Editor) 88
  Design mode 135
  Quick view mode 136
  WordPad 135
java.lang package 297
java.util package 335
JavaException exception 234
JCLASSES environment variable 139
Join transformation 16, 21, 35, 38, 50, 52, 69, 88

K

Key attribute 155, 413
  composite 157, 175, 202, 355
  foreign 216, 348, 350, 413
  identity relationships and 175
  single 155, 175, 357
Key attribute values
  checking for 248
  comparing 242
  retrieving as string 250
  setting 253
Keyboard shortcut 27
keysToString() method 250, 257

L

lastElement() method 338
logError() method 232, 407, 408
Logging 85, 408, 409

Logging *(continued)*
  example 408
  levels 408
  methods that send message 232, 407, 408
  principles of 408
  severity levels 408
Logging and tracing function block 113
Logging and Tracing/Log Error function block 113
Logging and Tracing/Log Information function block 114
Logging and Tracing/Log Warning function block 114
Logging and Tracing/Trace function block 115
Logical operator 257
logInfo() method 85, 232, 407, 408, 409
logWarning() method 232, 408
long data type 245, 251, 256, 285
LongText class
  determining data type 298
  getting attribute value 245
  obtaining maximum value 263, 264
  obtaining minimum value 265, 266, 267
  setting attribute 252
  valid conversions 300
LONGTEXT_TYPE constant 298
Lookup relationship 154, 188
  code for 191, 387, 388
  creating participant for 371
  defined 154, 176, 188
  defining 176, 188
  example of 154, 188
  participant type for 164, 177, 188
  relationship instance IDs 162
  static 133, 185
  testing 83

M

Maintain Composite Identity Relationship function block 203
maintainChildVerb() method 204, 213, 216, 352
  validations performed 353
maintainCompositeRelationship() method 354
  actions of 203
  deprecated version 391
  error messages 404, 405
maintainSimpleIdentityRelationship() method 356
  deprecated version 391
  error messages 404, 405
  validations performed 357
Managing child instances function blocks 207
Map definition 5, 7
  creating 29
  defined 5
  in map definition file 48
  loading 67
  location of 5
  naming conventions 5
  New Map wizard 29
  unloading 67
Map Designer 7, 13, 51
  Add Business Object dialog 32, 142
  Breakpoint dialog 77
  business object browser 17, 22, 24
  business object pane 17, 22, 33, 63, 143
  business object window 18, 24, 34, 141
  Context menu 25
  data conversion by 38
  Delete Business Object dialog 63
  Delete Map dialog 65
Map Designer *(continued)*
  exiting 24, 53
  files generated 11
  Find control pane 23, 49, 60, 61
  functionality of 23
  launching 14
  layout of 14
  main components 15
  main window 15, 21
  map workspace 18, 33, 143
  menus of 23
  Messages tab 18, 22, 403
  Multiple Attributes dialog 16, 38
  New Map wizard 29, 32
  Open file with map dialog 53
  Open Map from Project dialog 52
  output window 15, 19, 22, 23, 24, 71, 72
  overview 13
  preferences 19
  Programs toolbar 22, 23, 24, 168
  Save Map As dialog 32, 47
  search facility 60
  starting 14
  status bar 15, 22, 24, 170
  Submap dialog 44
  tab window 8, 51
  Tab window 15
  Test tab 18, 22, 72
  toolbars 22, 26, 170
  working in projects 14
Map development 10, 13
Map document 56, 60
Map execution
  continuing 78
  execution order 16, 52, 66, 69
  map instances and 7
  pausing 75, 78
  purpose of 146
  relationship instances and 160, 164
  test run and 72
  transactions and 228, 230
  viewing 72, 79
Map execution context 146
  calling context 146, 364, 365, 367, 368
  class for 146, 363
  cxExecCtx 146
  original-request business object 148, 196, 197, 201, 218, 220, 366
Map instance 7
  calling context 363, 366
  class for 227
  connector name 363, 366
  contents of 7
  defined 7
  execution context 7, 146
  original-request business object 365
  reusing 143, 144
  starting 410
  stopping 410
  trace level 411
  transaction programming model 231
Map properties 9, 54
  DataValidationLevel 85
  run-time 55
  Trace level 236, 410
  updating from server component management view 55, 144

Map Properties dialog (Map Designer)
- Business Objects tab 141, 143
- General tab 55, 228, 410

Map repository file 67
MapExeContext class 363, 369
- calling-context constants 147
- deprecated methods 368
- getConnName() 363
- getGenericBO() 368
- getInitiator() 363
- getLocale() 364
- getOriginalRequestBO() 365
- method summary 363
- setConnName() 366
- setInitiator() 366
- setLocale() 367

mapName._locale.txt message file 403
mapName.txt message file 405
Mapping
- defined 3
- overview 3
- simple 5
- standards 50, 86
- support for 3
- tools for 7, 8

Mapping API
- BusObjArray class 259
- CwDBConnection class 273
- CwDBStoredProcedureParam class 285
- DtpConnection class 289
- DtpDataConversion class 297
- DtpDate class 307
- DtpMapService class 333
- DtpSplitString class 335
- DtpUtils class 341
- IdentityRelationship class 345
- MapExeContext class 363
- Participant class 369
- Relationship class 377
- UserStoredProcedureParam class 393

Mapping function block 116
Mapping role 55
Maps
- base class for 227
- closing 53
- coding 87
- compiling 15, 18, 48, 70, 71, 73
- creating 28
- current 47, 51, 70, 229
- debugging 79, 85
- defined 3, 7, 13
- deleting 64
- development files 11
- exceptions and 144
- execution context 146
- HTML version 56
- improving modularity of 42
- map documents 56, 60
- name of 5, 32, 55, 229
- naming 32
- opening 51
- printing 62
- renaming 48
- saving 18, 32, 47, 63
- saving to file 49
- saving to project 47
- testing 72, 79

Maps *(continued)*
- validating 20, 47, 51, 69
- viewing execution 72, 79
- working with 51
- XML version 48

Maps function block 106
Maps/Constants function block 104
Maps/Exception function block 105
Math function block 116
MAX_CONNECTION_POOLS configuratin parameter 180, 183
max() method 262
maxBusObjArray() method 263
maxBusObjs() method 264
MaxLength attribute property 413
Message 18
- 5000 404
- 5001 404
- 5002 405
- 5003 405
- 5007 405
- 5008 405
- 5009 405
- format 405
- location of 18, 403
- number 406
- parameters in 403, 406
- revising 409
- severity 408
- text 406

Message file 403, 410
- choosing which one to use 403
- comments 407
- CWMapMessages.txt 404
- defined 403
- displaying 18
- format 405
- location of 12, 403
- maintaining 407
- mapName_locale.txt 403
- mapName.txt 405
- operations that use 407
- overview 403
- UserMapMessages.txt 404, 405
- using 406, 408

min() method 265
minBusObjArray() method 266
minBusObjs() method 267
Move transformation 16, 21, 35, 37, 50, 52, 69
Multiple-map map table 57

N

Name attribute property 413
Naming conventions
- maps 5
- participant definitions 164, 173
- relationship definitions 160, 173

New Constant 90, 94, 130
nextElement() method 338
nextRow() method (CwDBConnection) 281
nextRow() method (DtpConnection) 294
Non-identity relationships 154
NOT operator 257
not() 257
Null attribute value 249
Numbers, truncating 343

O

Object class 245, 251, 256
ObjectEventId attribute 50, 69, 74, 80
ObjectException exception 234
OKTOCONVERT constant 299
OperationException exception 234
Original-request business object 148, 196, 197, 201, 218, 220, 365
OUT parameter 287
Outbound map 3, 5
  example of customizing 212
  foreign key lookup in 349, 351
  in map document 57
  lookup relationship in 388
  testing 82, 84

P

Package
  importing Java packages 136
  java.lang 297
  java.util 335
padLeft() method 341
padRight() method 341
PARAM_IN constant 287
PARAM_INOUT constant 287
PARAM_OUT constant 287
Parent/child relationship 208
  adding child instance 345, 358
  defined 208
  defining 176
  deleting child instance 347, 358
Participant class 162, 165, 369, 375
  defined 369
  getInstanceId() 371
  getParticipantDefinition() 372
  getRelationshipDefinition() 372
  method summary 369
  Participant() 369
  set() 373
  setInstanceId() 373
  setParticipantDefinition() 374
  setRelationshipDefinition() 374
Participant definition 163
  advanced settings 175, 180
  copying 178, 179
  creating 173
  defined 163
  location of 163
  name of 372, 374
  naming conventions 164, 173
  renaming 179
Participant function block 107
Participant instance 164
  adding to relationship instance 378
  class for 165, 369
  constructor for 369
  contents of 165
  creating 369, 380
  data 165, 181, 369, 371, 373
  deactivating 381, 382
  defined 164, 369
  deleting 383, 384
  identifier 164
  participant definition 165, 369, 372, 374
  relationship definition 165, 369, 372, 374
Participant instance *(continued)*
  relationship instance ID 165, 369, 371, 373, 386
  retrieving from relationship instance 388
  updating 389
Participant instance identifier 164
Participant type 164, 173
  business object 164, 173, 174, 191, 202
  Data 154, 164, 173, 177, 188
Participant Types window 170, 171, 173, 177
Participant/Array function block 106
Participant/Constants function block 107
Participant() constructor 369, 374
Participants 163, 165
  defined 153
  naming conventions 164, 173
PATH environment variable 10, 70
POTENTIALDATALOSS constant 299
Preferences dialog (Map Designer) 25
  General tab 16, 20, 48, 51, 65, 67, 70, 141
  Key Mapping tab 21, 38, 39, 40, 44, 46
  Validation tab 21
prevElement() method 339
Project 14, 168
  browsing a 169
  opening a map from 14
  saving map to 47
  saving the map in 14
  working in 14
  working with 168
Properties function block 118

Q

Quick view mode (Activity Editor) 92

R

Relationship attribute property 413
Relationship class 161, 162, 377, 393
  addParticipant() 378
  create() 380
  deactivateParticipant() 381
  deactivateParticipantByInstance() 382
  defined 377
  deleteParticipant() 383
  deleteParticipantByInstance() 384
  deprecated methods 390
  getNewID) 385
  guidelines 377
  method summary 377
  retrieveInstances() 386
  retrieveParticipants() 388
  updateParticipant() 389
  updateParticipantByInstance() 389
Relationship database 160
  connecting to 230
  determining if transaction is in progress 294
  disconnecting from 235
  location of 11, 160, 161, 180, 182, 183
  queries for more rows to process 293
  retrieving next row 294
  rows affected by last write 293
  SQL queries 291
  type of 180, 182
  user account for 179, 180, 182
Relationship definition 159, 160

Relationship definition *(continued)*
  advanced settings 175, 179
  changing 174
  copying 178
  creating 173
  defined 7, 159, 167
  deleting 183
  dependent objects 223
  identity 174, 176, 191, 201, 202
  list 169
  loading 222
  location of 159
  lookup 176, 188
  name of 372, 374
  naming conventions 160, 173
  parent/child 208
  renaming 179
  saving 174
  unloading 221
  viewing 169
Relationship Designer 7
  Advanced Settings dialog 175, 179, 182, 184, 185
  Edit menu 172
  File menu 171
  functionality of 171
  Global Default Settings dialog 183
  Help menu 172
  launching 167
  layout of 169
  main window 170
  menus of 171
  overview 167
  starting 167
  status bar 172
  toolbar 172
  Tools menu 172
  View menu 172
  working with projects 168
Relationship development 165
Relationship function block 109, 118, 187
  Static Lookup 134
Relationship instance 160, 163
  adding a participant to 378
  class for 161, 345, 377
  creating 380
  creating participant for 370
  deactivating participant 381, 382
  defined 160
  deleting child objects 347
  deleting participant 383, 384
  location of 161
  retrieving instance ID 385, 386
  retrieving participants from 388
  run-time data 178
  updating participants 389
Relationship instance ID 162
  deactivating participant by 383
  defined 162
  deleting participant by 385
  identity relationship and 162
  in participant instance 371, 373
  lookup relationship and 162
  retrieving for participant 386
  retrieving next 385
  updating participant by 390
Relationship Manager 191
Relationship repository file 221
Relationship tables 160, 161
  caching 184
  changes to 149
  composite identity relationships 203
  contents of 163
  creating 175, 178, 181
  defined 161
  foreign 216, 349, 351
  foreign key lookups and 348, 350
  identity relationships 191
  index size 203
  location of 160, 161, 180, 182, 183, 184
  lookup relationships 133, 176, 177, 189
  MaxLength attribute 203
  modifying 350
  name of 161, 181, 189
  participants in 382, 383
  performing lookup in 211
  table schemas 8, 178, 179, 182
RelationshipRuntimeException class 82, 145, 404
Relationships 159, 163
  defined 153
  dynamic 184
  exceptions 145
  implementing code for 187
  introduction 153, 166
  naming conventions 160, 173
  non-identity 154
  optimizing 184
  starting 174
  static 133, 184
  stopping 174
  testing 80
  transformations for 187
  types of 153, 180
  working with 187
release() method 281
releaseRelConnection() method (deprecated) 235
removeAllElements() method 268
removeElement() method 268
removeElementAt() method 269
Replace text 62
repos_copy utility 67, 221
Repository
  exporting a map from 67
  exporting a relationship 221
  relationship database and 160, 161
Required attribute 250, 413
reset() method 340
Retrieve verb 355, 357
  foreignKeyXref() and 218, 220
  maintainChildVerb() and 214, 215
  maintainCompositeRelationship() and 204
  maintainSimpleIdentityRelationship() and 194, 196, 197, 200
retrieveInstances() method 211, 386
retrieveParticipants() method 211, 388
Retrieving
  business object array contents 262
  business object array maximum value 262, 263, 264
  business object array minimum value 265, 266, 267
  business object array values as string 271
  business object attribute 245
  business object from array 261
  business object key attribute value as string 250
  business object type 247
  business object verb 247

Retrieving *(continued)*
last index from business object array 262
map name 229
number of elements in business object array 270
rollBack() method (CwDBConnection) 282
rollBack() method (DtpConnection) 295
runMap() method 221, 333

S

SELECT statement 276, 277, 280, 281
Server component management view
updating map properties 55, 144
updating relationship properties 185
SERVICE_CALL_FAILURE calling context 147, 193
generic business object and 148
getOriginalRequestBO() and 365
maintainCompositeRelationship() and 204
maintainSimpleIdentityRelationship() and 200
original-request business object 148, 365
retrieving 364
setting to 367
SERVICE_CALL_REQUEST calling context 147, 193
Create verb and 195, 214, 220
Delete verb and 196, 214, 220
foreignKeyXref() and 219, 351
generic business object and 148
getOriginalRequestBO() and 365
maintainChildVerb() and 214, 354
maintainCompositeRelationship() and 204, 355
maintainSimpleIdentityRelationship() and 195, 357, 358
original-request business object 148, 365
Retrieve verb and 196, 214, 220
retrieving 364
setting to 367
testing with 82, 84
Update verb and 196, 214, 220
updateMyChildren() and 210
SERVICE_CALL_RESPONSE calling context 147, 193
Create verb and 197, 215, 218
Delete verb and 197, 215
foreignKeyXref() and 217, 351
generic business object and 82, 148
getOriginalRequestBO() and 365
identity relationships and 82
maintainChildVerb() and 214, 354
maintainCompositeRelationship() and 204, 355
maintainSimpleIdentityRelationship() and 197, 358
original-request business object 148, 199, 365
Retrieve verb and 197, 215, 218
retrieving 364
setting to 367
testing with 83, 85
Update verb and 197, 215, 218
updateMyChildren() and 210
ServiceCallException exception 234
Set Value transformation 16, 35, 36, 50, 52, 69, 88
set() method 251, 257, 373
set12MonthNames() method 328
set12MonthNamesToDefault() method 329
set12ShortMonthNames() method 329
set12ShortMonthNamesToDefault() method 329
set7DayNames() method 330
set7DayNamesToDefault() method 330
setConnName() method 366
setContent() method 252
setDefaultAttrValues() method 253
setElementAt() method 269
setInitiator() method 366
setInstanceId() method 373
setKeys() method 253
setLocale() method 367
setParamDataTypeJavaObj() method (deprecated) 398
setParamDataTypeJDBC() method (deprecated) 398
setParamIndex() method (deprecated) 399
setParamIOType() method (deprecated) 399
setParamName() method (deprecated) 400
setParamValue() method (deprecated) 400
setParticipantDefinition() method 374
setRelationshipDefinition() method 374
Setting
business object attribute 251, 254
business object attribute default value 253
business object contents 252
business object key attribute value 253
business object value in an array 269
business object verb 254
setVerb() method 254, 257
setWithCreate() method 254
Simple identity relationship 155, 156, 174, 191
child-level 201
defining 175, 176, 191, 201
defining Cross-Reference transformation 191
defining transformation rules 201
example of 155
main map 201
maintainChildVerb() 201, 216
maintainSimpleIdentityRelationship() 191, 356
parent map 201
participant type for 191
submap 202
Single-map map table 56
size() method 262, 270
Source business object 3, 5, 144
adding to map 29, 33
business object window 34
displaying 9, 17, 24, 55
testing 73
variable for 140
Split transformation 16, 21, 36, 40, 50, 52, 69, 88
Splitting strings
creating the parsed string 335
getting element at specified position 336
getting first element from string 336
getting last element from string 338
getting next element from string 338
getting number of elements in string 337
getting previous element from string 339
processing the parsed tokens into an object 338
resetting current position number 340
SQL query
checking for more rows 279, 293
executing 275, 276, 278, 291
prepared 275
retrieving next row 281
static 276
Standard toolbar (Activity Editor) 90
Copy 90
Cut 90
Delete 91
Help 91
Paste 91
Print Activity 90
Save Activity 90

Standard toolbar (Map Designer) 22
- displaying 22, 24
- Find 60
- New Map 29
- Open Map from File 53
- Open Map from Project 52
- Print 62
- Save Map to File 49
- Save Map to Project 47

Standard toolbar (Relationship Designer) 171
- displaying 170, 172
- New Participant 173
- New Relation 173
- Save Relation 174

start_server.bat file 140
Static lookup 211
Static Lookup relationship 133
Static relationship 184
Status bar (Activity Editor) 91
Stored procedure
- executing 276, 277, 278, 292
- for relationship instance 178, 181
- query result 280, 281

Stored-procedure parameter
- creating object for 285, 393
- in/out parameter type 286, 396, 399
- index position 395, 399
- Java Object type 394, 398
- JDBC data type 395, 398
- name 397, 400
- value 287, 397, 400

String class 413
- as stored-procedure parameter type 285
- checking for valid data 256
- converting to 305
- converting to Boolean 300, 303
- converting to Double 301, 303
- converting to Float 302, 304
- converting to Integer 302, 305
- determining data type 298
- getting attribute value 245
- obtaining maximum value 263, 264
- obtaining minimum value 265, 266, 267
- setting attribute to 251
- valid conversions 300

String function block 119
- Upper Case 123

STRING_TYPE constant 298
stringReplace() method 342
Strings
- padding with specified character 341
- replacing one pattern with another 342

StringTokenizer class 335
Submap transformation 36
Submaps 41
- accessing code for 88
- attribute comment for 50
- calling 43, 333
- child business objects 42, 44
- compiling 44, 70
- conditions 45
- creating 43
- defined 41
- identity relationships and 202
- key mapping for 21
- naming conventions 44
- transformation code for 16

Submaps *(continued)*
- uses for 41
- validating 52, 69

sum() method 270
swap() method 270
Switch to Project (Relationship Designer) 169
System Manager 9
- compiling a map 70
- Component menu 70, 174, 410
- Map Properties window 55, 144, 411
- opening map from project in 52
- relationship categories 184
- starting Map Designer from 14
- starting Relationship Designer from 167

SystemException exception 234

T

Table tab (Map Designer) 15, 17
- adding business object 33
- attribute transformation table 15, 63
- business object pane 17, 22, 24, 33, 63, 143
- business object variables 141
- calling a submap 44
- custom transformation 46
- default display 22
- deleting a transformation 63
- deleting business object 63
- joining attributes 38
- moving attribute 37
- output window 15
- setting attribute value 36
- specifying execution order 66
- splitting attribute 40
- temporary business object 143

Temporary variables 142
Test run 72
- breakpoints 75, 78
- creating test data 73
- initial 73
- pausing 75, 78
- preparing for 73
- starting 79
- subsequent 75
- viewing results 79

toBoolean() method 300
toDouble() method 301
toFloat() method 301
toInteger() method 302
Tools menu (Activity Editor) 90
- Translate 90

Tools menu (Map Designer) 25
Tools menu (Relationship Designer) 172
toPrimitiveBoolean() method 303
toPrimitiveDouble() method 303
toPrimitiveFloat() method 304
toPrimitiveInt() method 304
toString() method 255, 257, 271, 305, 330
Trace level 231, 409, 410
Trace message 236, 407, 409, 410
- adding 409
- assigning trace level to 409
- generating 410
- setting trace level for 410

trace() method 236, 407, 409
Tracing 409, 410
- code example 410

Tracing *(continued)*
generating message 410
level for 409
Transactions
beginning 273, 289
committing 274, 290
determining if in progress 280, 294
rolling back 282, 295
Transformation code
deleting 63
finding text in 60
handling exceptions in 144
location of 70
missing 49
viewing 60
Transformation step 5, 16, 28, 34, 63, 69, 87
Transformations 6, 16, 35
addressing attributes 140
checking completeness of 49
coding 87
Context menu 25
Cross-Reference 16, 36, 45
Custom 16, 36, 46
defining for relationships 187, 204
destination attribute 16
execution order 16, 52, 66, 69
in map definition file 48
introduction 5
Join 16, 35, 38
map document for 56
Move 16, 35, 37
relationship attributes 187
selecting 24
Set Value 16, 35, 36
source attribute 16
Split 16, 36, 40
standard 16, 35, 88
Submap 16, 36
validating 51, 69
validating source data 146
variables 142
truncate() method 343
Type attribute property 413

U

UNKNOWN_TYPE constant 298
UPDATE statement 276, 277, 279
Update verb
conditionally set 211
foreignKeyXref() and 218, 220
maintainChildVerb() and 214, 215
maintainCompositeRelationship() and 204
maintainSimpleIdentityRelationship() and 194, 196, 197, 200
updateMyChildren() method 210, 358, 391
updateParticipant() method 389
updateParticipantByInstance() method 389
UserMapMessages.txt message file 404, 405
UserStoredProcedureParam class (deprecated) 393, 400
constructor 393
getParamDataTypeJavaObj() 394
getParamDataTypeJDBC() 395
getParamIndex() 395
getParamIOType() 396
getParamName() 397
UserStoredProcedureParam class (deprecated) *(continued)*
getParamValue() 397
method summary 393
setParamDataTypeJavaObj() 398
setParamDataTypeJDBC() 398
setParamIndex() 399
setParamIOType() 399
setParamName() 400
setParamValue() 400
UserStoredProcedureParam() constructor (deprecated) 393
Utilities function block 121
Utilities/Vector function block 121

V

validData() method 256
Variable 140
cwExecCtx 146, 333, 346, 347, 349, 350, 355, 357, 359
for business object 140
global 142
temporary 142
Vector class
with executeStoredProcedure() 276, 286
with nextRow() 281
Verb
defined 35
retrieving 247
setting 35, 50, 210, 254, 352
test run 73, 75
View menu (Activity Editor) 89
Content window 89
Design mode 89
GoTo 89
Library window 89
Preferences 90
Properties window 89
Quick view mode 89
Status Bar 90
Toolbars 89
Zoom In 89
Zoom Out 89
Zoom To 89
View menu (Map Designer) 22, 24
Business Object Pane 17, 22, 24
Clear Output 15, 22, 24, 71
Diagram 18, 22, 24, 36
Output Window 15, 22, 24
Preferences 19, 25
Server Pane 18, 22, 24
Status Bar 15, 22, 24
Toolbars 22, 24
View menu (Relationship Designer) 170, 172
Collapse Tree 172
Expand Tree 172
Participant Types 172, 173
Status Bar 170
Toolbar 170

W

Warning message 232, 408

Z

Zero-length string 248