IBM VisualAge® for Java™, Version 3.5

XMI Toolkit

Note!

Before using this information and the product it supports, be sure to read the general information under **Notices**.

Edition notice

This edition applies to Version 3.5 of IBM VisualAge for Java and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

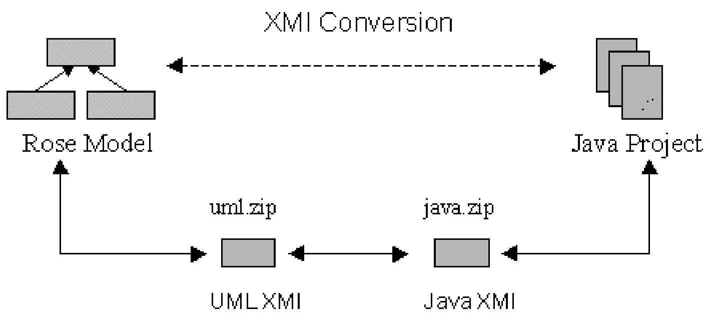
Chapter 1. XMI Toolkit - overview

The XMI Toolkit for VisualAge for Java utilizes the XML-based standard, XMI, to perform transformations on different forms of programming and modeling information. XMI is an open standard for general purpose information interchange and messaging. XMI can be applied to a wide variety of information exchanges and tranformations. This XMI Toolkit implementation specializes in interchange of UML design information and Java source code. The Toolkit consists of a graphical user interface and command line programs that are used to perform these conversions. Additionally, given an XMI document, the XMI Toolkit can generate a corresponding DTD.

For example, the XMI Toolkit for VisualAge for Java allows you to develop applications by jointly using Rational Rose and IBM VisualAge for Java. In a typical scenario, you first develop an object-oriented analysis model for your application using the Rational Rose visual modeling tool. Next, you use the XMI Toolkit to generate Java code corresponding to the model. The generated Java code can then be imported into IBM VisualAge for Java, where the implementation of the application's class methods can be completed.

During the development phase in VisualAge for Java, it may be necessary to make structural modifications to the original analysis model you created using Rose. For example, you might need to add or delete some classes, methods, or fields, or to change the structure of the class inheritance hierarchy. The XMI Toolkit provides the capability to create an updated version of the analysis model that reflects the changes made during development in VisualAge for Java. In this way, the analysis model can be kept in synch with the application source code.

The diagram below shows the steps involved in the conversion process between a Rose model and the generated Java source. The broken line in the upper portion of the diagram shows how you can conceptually envision the conversion process between a Rose analysis model and the corresponding Java source files in a Java project. The solid lines in the lower portion of the diagram provide a more detailed flow of how the XMI conversion is actually carried out.



The steps are as follows:

1. The Rose model is converted into one or more XMI files that contain the UML XMI representation of the model's structure. The UML XMI files are packaged

into a single file named uml.zip. This file is written to the same directory where the corresponding Rose .mdl file is stored.
2. The XMI Toolkit then maps the UML XMI representation into an equivalent representation in Java XMI. The corresponding Java XMI files are stored in a single file named java.zip. This file is written to the top level directory ("project directory") under which the Java project source files are to be written.
3. Finally, the XMI Toolkit generates the Java source files corresponding to the representation in Java XMI.

The process also works in reverse. If you have already implemented an application in Java, you can use the XMI Toolkit to derive a corresponding Rose model. You can then work with the generated model file to document or enhance the application's design.

Important! When roundtripping between VisualAge for Java and Rational Rose, you must specify the same locations (directory and .mdl file name for Rose, and Java project directory for Java) for the input and output fields when going back and forth. The file names must stay the same; consequently, the files will be overwritten each time conversion occurs. This is required since the XMI information in the uml.zip and java.zip files written during the previous conversion is needed to perform the correct transformation on the conversion back.

Therefore, we recommend that you always keep a backup copy of your model (i.e. the .mdl and .cat files).

RELATED CONCEPTS

The XMI standard
XMI Toolkit Browser and SmartGuide

RELATED TASKS

Performing an XMI conversion
Going from a Rose Model to Java
Going from Java to a Rose Model
Performing XMI conversions at the command line

The XMI standard

The XML Metadata Interchange Format (XMI) specifies an open information interchange model that is intended to give developers working with object technology the capability to exchange programming data over the Internet in a standardized way.

XMI is an accepted industry standard that combines the benefits of the Web-based XML standard for defining, validating, and sharing document formats on the Web with the benefits of the object-oriented Unified Modeling Language (UML). UML itself is a specification of the Object Management Group (OMG) that provides application developers with a common language for specifying, visualizing, constructing, and documenting distributed objects and business models.

By using an industry standard for storing and sharing object programming information, development teams using tools from multiple vendors can collaborate on applications. The XMI standard will allow developers to leverage the Web to exchange object-oriented data among tools, applications, and repositories, and to create secure, distributed applications built in a team development environment.

For more information about XMI, see the IBM XMI Web page at http://www.software.ibm.com/ad/features/xmi.html.

RELATED CONCEPTS

XMI Toolkit - overview
XMI Toolkit Browser and SmartGuide

XMI Toolkit Browser and SmartGuide

The XMI Toolkit user interface consists of two components:

- The XMI Toolkit Browser and
- The XMI Tookit SmartGuide

You use the XMI Toolkit Browser to examine the XMI conversion mapping between a Rose model and a Java project. The Browser displays the UML XMI representation of your Rose model in the uml.zip file corresponding to that model; it displays the Java XMI representation of a Java project in the java.zip file corresponding to that project. The Browser can also be used to show differences between successive versions of the UML XMI representation of a Rose model, or the Java XMI representation of a Java project.

The XMI Toolkit SmartGuide is launched from the Browser whenever you initiate an XMI conversion between a Rose model or Java project. The SmartGuide provides a sequence of easy-to-use panels that assist you in organizing and specifying the details of the conversion you want to perform.

The XMI Toolkit Browser
Using the XMI Toolkit Browser view, you can:

1. Launch the XMI Toolkit SmartGuide to perform an XMI conversion between a Rose model and a Java project.
2. Examine the XMI mapping between a Rose model and a Java project.
3. Compare the differences between two versions of the UML XMI representation of a Rose model.
4. Compare the differences between two versions of the Java XMI representation of a Java project.
5. Generate a DTD.

The Toolkit Browser contains a side-by-side display of two tree views. Each tree view displays the hierarchical structure of either the UML XMI representation of a Rose model or the Java XMI representation of a Java project. You use both tree views together either to examine the mapping between a Rose model and its corresponding Java project, or to compare the differences between the two most recent versions of either UML XMI or Java XMI.

When you use the Browser to view an XMI conversion mapping between Rose and Java, one tree displays the Rose model in UML XMI, while the other tree displays the Java project in Java XMI. The tree on the left always contains the XMI representation of the input to the conversion, and the tree on the right contains the XMI representation of the output.

When you are viewing differences between the UML XMI representation of a Rose model with the previous version, both trees display UML XMI information. Additionally, a panel is displayed beneath the two tree views that contains a table of the differences between the two versions of the UML XMI. The leftmost tree always displays the newer version of the UML XMI. It reflects the XMI information contained in the uml.zip file located in the Rose model directory. The rightmost tree displays XMI information contained in the previous version of the uml.zip file, which is named uml_old.zip.

Similarly, when you are viewing differences between the Java XMI representation of a Java project with the previous version, both trees display Java XMI information. The panel beneath the two tree views contains a table of the differences between the two versions of the Java XMI. The leftmost tree always displays the newer version of the Java XMI. It reflects XMI information contained in the java.zip file stored in the Java project directory. The rightmost tree displays XMI information contained in the previous version of the java.zip file, which is named java_old.zip.

The Browser menubar

The Browser menubar has the following four menu choices:

- File
- View
- Window
- Help

The File menu is used to perform an action within the Toolkit. From the File menu, you can select menu items to:

- Perform an XMI conversion between Rose and Java
- Show the UML/Java mapping created during an XMI conversion between Rose and Java
- Show the differences between two versions of UML XMI
- Show the differences between two versions of Java XMI
- Generate a DTD
- Set the Browser preferences
- Exit the Browser

The View menu contains selections with which you can specify the size and location of the Browser window panes. Additionally, the view menu contains a filter that you can use to specify what kinds of differences you want to view when comparing two different versions of XMI. Specifically, from the View menu, you can select menu items to:

- Maximize or minimize the selected Browser pane.
- Detach the selected pane from the Browser.
- Return a detached pane to the Browser (from the View menu of the detached pane).
- Set the differences filter to include or exclude categories of differences when comparing two versions of XMI.
- Expand or collapse trees in the Browser tree views.

When you detach a pane from the Browser, the View menu for the detached pane will have a Return to the parent page menu item. You use this to return the detached pane to its former position in the Toolkit Browser.

The Windows menu contains a list of the windows associated with the XMI Toolkit. The list includes the Browser itself, and any panes that have been detached from the Browser.

The Help menu contains an item to view general information about the XMI Toolkit.

Setting preferences
You can use the Application Preferences dialog to control certain aspects of the XMI Toolkit appearance and behavior. Use the **File > Preferences** menu item to bring up the Application Preferences dialog.

The Application Preferences dialog has the following areas:

- **General** Allows you to set preferences on the appearance of the panes in the Browser. For example, you can specify if you want title bars shown with the panes, or if you want the active pane to be highlighted.
- **Appearance** Allows you to control the general look-and-feel of the Browser by selecting from several Family and Platform options.
- **Toolbar** Allows you to specify the positioning and behavior of the Browser toolbar. Additionally, you can specify aspects of the appearance of the toolbar icons.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Starting the Browser
Performing an XMI Conversion
Going from a Rose Model to Java
Going from Java to a Rose Model
Examining an XMI Mapping between Rose and Java
Viewing Differences between Versions of XMI files
Generating a DTD

Chapter 2. Using the XMI Toolkit

Launching the XMI Toolkit Browser

You can launch the XMI Toolkit Browser either from the VisualAge for Java IDE, or from a command prompt. **Important!** JDK 1.2.2 must be installed on your system in order for the XMI Toolkit to run.

To launch the XMI Toolkit Browser from VisualAge for Java, select **Workspace** > **Tools** > **XMI Toolkit**.

To launch the XMI Toolkit Browser from a command prompt, enter `xmitk`.

When you launch the Browser, the XMI Toolkit - Console window appears. This window is used to display progress information and error messages that are not displayed within the Browser.

RELATED CONCEPTS

XMI Toolkit - overview
XMI Toolkit Browser and SmartGuide

RELATED TASKS

Going from a Rose model to Java
Going from Java to a Rose model
Examining an XMI mapping between Rose and Java
Viewing differences between versions of XMI files

Performing an XMI conversion

To perform an XMI conversion between a Rose model and a Java project, launch the XMI Toolkit SmartGuide by selecting **File > Perform a Conversion** from the XMI Toolkit Browser menubar.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Launching the XMI Toolkit
Going from a Rose model to Java
Going from Java to a Rose model
Performing XMI conversions at the command line
Sample: Rountripping between Rational Rose and VisualAge for Java - Part I

Going from a Rose model to Java

You can use the XMI Toolkit to convert a model developed in Rational Rose into a Java project by generating Java source files. You must adhere to the following when creating Rose models that you intend to convert into Java using the XMI Toolkit:

- Develop your models using only Rational Rose 98i, Enterprise Edition or Rose 2000, Enterprise Edition.

- Develop your models with the Default Language set to Java. Before starting your model, select the **Tools > Model Properties > Edit** menu item in Rational Rose. When the **Options** notebook appears, click the **Notation** tab, and select **Java** as the **Default Language**. Click **OK** to set the default language and close the notebook.
- Do not use the frameworks provided with Rational Rose.
- Perform a Java syntax check on the model before you attempt to convert it to Java, and correct any errors or warnings that are detected. To invoke a Java syntax check in Rational Rose, select **Tools** > **Java** > **Syntax Check**. **Attention:** You may bridge models that have warnings about unresolved types. You will need to use import statements in Java to resolve the types.

Only the contents of the Rose Logical view are used in the conversion to Java. In this version, the following modeling constructs are supported:
- Package
- Class
- Interface
- Attribute
- Operation
- Generalization Link
- Association Link
- Realization Link

When carrying out a Rose to Java conversion, use the XMI Toolkit SmartGuide to specify the details of the conversion. To bring up the SmartGuide, select **Tools** >**Perform a Conversion** from the XMI Toolkit Browser menu item.

For a Rose to Java conversion, click the arrow so that it is pointing right. Complete the following:
1. Specify the location of your Rose model in the **Rational Rose Model** entry field. Either enter the complete path name of the .mdl file, or click **Browse** to navigate to its location on the file system.
2. In the **Java Project Directory** entry field, specify a project directory where you want your Java files generated. You may either type the full path name directly into the entry field, or use the **Browse** button to navigate to the location on the file system. During the Java code generation, subdirectories that mirror your model's package hierarchy will be generated beneath the project directory. The generated .java source files will reside in the appropriate subdirectories according to their package statements.
   Note: If the project directory you specify contains any .java or .class files prior to the conversion, they will automatically be backed up for you. For example, if the name of the project directory you specified is D:\mydir, the .java (and .class) files will be moved under D:\mydir_old (in directories corresponding to their package structure) prior to generating the new .java files. This ensures that the .java files contained in your project directory include only those from the most recent conversion from Rose. Any existing .java (and .class) files in the D:\mydir_old directory are deleted prior to moving the ones in the D:\mydir directory. In this way, after the conversion, the D:\mydir directory contains only the latest .java files, and the D:\mydir_old directory contains the .java (and .class) files that were in D:\mydir prior to performing the conversion.

For convenience, the SmartGuide remembers the previous five locations you specified for the input and output of a conversion. The SmartGuide will automatically fill in the last location you used for input or output as the default.

The other remembered locations can be selected from the pull-down list associated with the entry field where you specify the input and output locations.

After specifying the location of your input and output, click **Next** to bring up the Input Options page of the SmartGuide.

The Input Options page contains a button that launches a dialog to specify virtual paths for your Rose model. Click the **Path Mapping** button to launch the Path Mapping dialog; use this dialog to associate symbols with actual paths. This allows you to convert Rose models that have controlled units that use virtual paths.

After making your selections for the Input Options, click **Next** to go to the Output Options page of the SmartGuide. In the Output Options page, use the radio buttons to specify code generation options for association roles and class attributes.

Association roles become fields within a Java class during a Rose to Java conversion. If the association role has a multiplicity greater than 1, the type of the corresponding field in the Java class will be either an array of the associated class or a Vector. Select either **Vector** (the default) or **Array**.

Use the checkboxes to indicate if you want accessor methods generated for the association roles or class attributes. The default is *not* to generate accessors. For non-boolean attributes, accessor methods will be generated to get and set the generated Java field. For a boolean attribute, a method will be generated to set the field, along with a method to check the true/false value of the field. For association roles with multiplicity greater than 1, an accessor will be generated to get the array or Vector representing the association. Additionally, accessors will be generated to add elements to the Vector or array, and to remove elements from it. A field initializer will be generated for a Vector or array that corresponds to an association role with a multiplicity greater than 1.

The table below summarizes the accessors that are generated:

Accessor Prefix	**Parameters**	**Return Value**	**Example**	**Notes**
get	None	field value (non-boolean fields)	getBalance()	For all non-boolean fields, including those designating association roles
is	None	field value (boolean fields)	isReady()	Boolean fields only
set	new field value	None	setBalance(float balance)	For all fields, except: (a) those designating association roles with multiplicity > 1 and (b) attributes that are final
add	new element	None	addAcctRole(Acct item)	For fields designating association roles with multiplicity > 1 only

remove	index of element to remove	element removed	removeAcctRole(int index)	For fields designating association roles with multiplicity > 1 only

For enhanced readability, the generated accessors follow the convention of upper-casing all words in the accessor name, except for the first. Thus, getBalance() would be generated for a field called “balance.” Because of this convention, you should begin all your UML attributes and role names with a lower-case letter. Although you can begin them with upper-case letters, if you had both a field named ″balance″ and another named ″Balance″ in the same class, each would have a generated accessor named ″getBalance()″, which would cause a Java compiler error.

You can specify import statements to be included in your code using the multi-line entry field provided on the Output Options page of the SmartGuide. Enter the import statements that you would like included in your source files (for example, import java.util.*;). Be sure the **Include the following imports** checkbox is checked. For any class which has a Java source file from a previous version, the imports you enter will be added to those already in the previous version (if not already there).

The XMI Toolkit will remember the import statements you have entered across sessions, so you will not need to re-enter them each time you generate new Java source files. You may uncheck the **Include the following imports** checkbox if you decide not to include the imports during a Rose to Java conversion, or because you need them only in some source files and will enter them yourself. That way, you will not have to re-enter them later for a Java project that requires the import statements. All you will need to do is check the **Include the following imports** checkbox, and the imports will again be added to your project’s generated Java files.

After specifying the output options, click **Finish** to start the Rose to Java conversion. When the conversion starts, a progress indicator will appear near the upper left portion of the XMI Toolkit Browser. This will allow you to monitor the progress in each phase of the conversion. A message box will appear when the conversion is complete. Click **OK** to close the message box.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I
Examining an XMI mapping between Rose and Java
Going from Java to a Rose model
Performing XMI conversions at the command line

RELATED REFERENCES

UML to Java mapping
Java to UML mapping

Going from Java to a Rose model

You can use the XMI Toolkit to convert a Java project developed in VisualAge for Java into a Rose model. To convert a Java project into a Rose model, the following conditions must be met:

- All source files in your Java project must be successfully compiled.
- Both the Java source and compiled class files must reside on the file system.

Before converting a Java project into a Rose model, you must first export the Java source and class files for the project from the VisualAge for Java IDE. **Note:** You must correct any compilation errors in your project before exporting it. To export your project, select it in the VisualAge for Java IDE Workbench, right-click to bring up the pop-up menu, and select the **Export** menu item. When the Export SmartGuide appears, ensure that the **Directory** radio button has been selected, and click **Next**. On the next page, enter the directory where you would like to export your Java project. Ensure that both the **.java** and **.class** checkboxes are checked. Click **Finish** to export your project.

Note: When exporting your project from VisualAge to a Java project directory used in a previous conversion, allow VisualAge to overwrite all the .java and .class files that reside beneath the project directory. If, however, you have deleted or renamed classes in VisualAge, you should delete the old .java and .class files so that they are are not included when bridging back to Rose. You should not, however, delete the java.zip file from your last conversion from Rose, since it will be used during the conversion back to Rose.

Complete the following:

1. As with a Rose to Java conversion, use the XMI Toolkit SmartGuide to specify the details of a Java to Rose conversion. To launch the SmartGuide, select the **Tools** > **Perform a Conversion** menu item in the Toolkit Browser.
2. When the SmartGuide appears, in the **Java Project Directory** entry field, type the full path name of the project directory under which your Java .class files reside (i.e. the directory to which you exported your project from VisualAge for Java). You may use the **Browse** button to navigate to the directory on the file system instead of typing the directory path name into the entry field.
3. In the **Rational Rose Model** entry field, enter the full path name of the Rose model you would like to create. You can use the **Browse** button to navigate to the location of the Rose .mdl file if the .mdl file already exists.
4. Ensure that the arrow is pointing left. You can change the arrow's direction by clicking it.
5. **Important!** If you have performed a Rose to Java conversion, and are now converting Java back to Rose, you must use the same path and file name for the .mdl file as you used before. Since this will overwrite your Rose model, be sure to create a backup copy of your Rose model (.mdl and .cat) files.
6. After specifying the input and output types and locations, click **Next** to go to the Input Options page in the SmartGuide.
7. In the Input Options page, you can select which packages in your Java project you would like to include in your Rational Rose model. If you have just one package in your project, it will be selected by default. If you have more than one package, none will be selected. You can use the checkbox next to each package to select or deselect that package. Use the **Select All** and **Deselect All** buttons to select or deselect all the packages at once. When you have selected which packages to include in your Rose model, click **Next** to go to the Output Options page of the SmartGuide.

8. In the Output Options page, you can specify a virtual path mapping for your Rose model .cat files. Click the **Path Mapping** button to bring up the dialog to map symbols to actual paths.
9. After specifying the output options, click **Finish** to start the Java to Rose conversion. When the conversion starts, a progress indicator will appear near the upper left portion of the XMI Toolkit Browser. This will allow you to monitor the progress in each phase of the conversion. A message box will appear when the conversion is complete. Click **OK** to close the message box.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I
Examining an XMI mapping between Rose and Java
Going from a Rose model to Java
Performing XMI conversions at the command line

RELATED REFERENCES

Java to UML mapping
UML to Java mapping

Examining an XMI mapping between Rose and Java

After performing a conversion between Rose and Java, you will automatically be shown the XMI conversion mapping in the two tree views in the Browser unless you have unchecked the **Display Conversion Mapping** checkbox on the first page of the SmartGuide. The input will appear in the left tree view, and the output will appear in the right tree view. You can examine the XMI mapping between the UML XMI information and the Java XMI information by clicking a node in one of the trees and observing the corresponding node which is then highlighted in the other tree.

You can also view a mapping from a different XMI conversion done previously, by selecting **File** >**Show Rose - Java Mapping** menu item. A dialog opens.

Using this dialog, first select the **uml.zip** file containing the UML XMI representation of a Rose model. Next, select the **java.zip** file containing the Java XMI representation of the corresponding Java project. For convenience, the XMI Toolkit remembers the location of the java.zip and uml.zip files used in the last conversion. After selecting the two files, click **OK**. The XMI Toolkit loads the UML XMI information into the the tree view on the left, and the Java XMI information into the tree view on the right.

You can examine the XMI mapping between the UML XMI information and the Java XMI information by clicking a node in one of the trees and observing the corresponding node which is then highlighted in the other tree.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Going from a Rose model to Java
Going from Java to a Rose model
Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I

RELATED REFERENCES

UML to Java mapping
Java to UML mapping

Viewing differences between versions of XMI files

Since updates made in VisualAge for Java may affect the contents and organization of your Rose model, you may wish to compare the current UML XMI information for your model with the previous version. Similarly, changes made in Rose may affect the contents of the corresponding Java project. You can see how changes made in Rose affect your Java project by comparing the current Java XMI information with the previous version.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Viewing differences between UML XMI files
Viewing differences between Java XMI files
Going from a Rose model to Java
Going from Java to a Rose model
Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I

Viewing differences between UML XMI files

To compare UML XMI information with the previous version, select **File** > **Show UML-UML Differences**. A file dialog opens. Use this dialog to navigate to the directory containing your Rose model. Select the **uml.zip** file written there by the XMI Toolkit during the previous XMI conversion, and then click **Open**.

The XMI Toolkit loads the uml.zip file into the tree view on the left, and the previous version, uml_old.zip, into the tree view on the right. The differences panel in the lower part of the browser displays a tabular representation of the differences between the two versions of the UML XMI information. Clicking an entry in the differences table will highlight the corresponding nodes in the tree views. Constructs that have been added will be highlighted in the tree on the left (the tree displaying the current UML XMI information for your model). Constructs that have been deleted will be highlighted in the tree on the right (the tree displaying the UML XMI information for the previous version of your model). Entries for added and deleted links (association, generalization, and realization) are included in the table, but they do not highlight in the trees when clicked.

Tip: You may alphabetically sort any of the four columns in the differences panel by single-clicking on the column heading. To sort in reverse order, click the column heading while holding down the Shift key.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I
Going from a Rose model to Java
Going from Java to a Rose model
Viewing differences between Java XMI files

Viewing differences between Java XMI files

To compare Java XMI information with the previous version, select **File** > **Show Java-Java Differences**. A file dialog opens. Use this dialog to navigate to the root directory of your Java project source files. Select the **java.zip** file written there by the XMI Toolkit during the last XMI Conversion, and then click **Open**.

The XMI Toolkit loads the java.zip file into the tree view on the left, and the previous version, java_old.zip, into the tree view on the right. The differences panel in the lower part of the browser will show a tabular representation of the differences between the two versions of the Java XMI information. Clicking an entry in the differences table will highlight the corresponding nodes in each of the trees. Constructs that have been added will be highlighted in the tree on the left (the tree displaying the current Java XMI information for your model). Constructs that have been deleted will be highlighted in the tree on the right (the tree displaying the Java XMI information for the previous version of your model).

Because constructs in Java are identified uniquely by name, changing the name of a construct will cause it to appear as both an add and a delete difference when viewing Java-Java differences. For example, if you have an attribute called a1, and you rename it to a2 in the Java source file, in the Java-Java differences, it will show that an attribute a1 was deleted, and that a2 was added.

Methods are uniquely identified by their name and parameter types. Therefore, if you change the name of a method, add or delete a parameter, or change the type of parameter, the method and all its parameters will be treated as additions to the new model in the Java-Java differences. The previous method and parameters will be treated as deleted from the old model in the Java-Java differences.

Tip: You may alphabetically sort any of the four columns in the differences panel by clicking the column heading. To sort in reverse order, click the column heading while holding down the Shift key.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I
Sample: Roundtripping between Rational Rose and VisualAge for Java - Part II
Viewing differences between UML XMI files
Going from a Rose model to Java
Going from Java to a Rose model

Generating a DTD

To generate a DTD for a UML XMI model, select **File** > **Generate DTD**. A dialog opens allowing you to specify the file name and location for the generated DTD. Use the dialog to navigate to the directory where you want the DTD to be generated. You must enter the DTD name in the **File name** entry field. Click **Save** to generate the DTD.

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

DTD generation conventions

Performing XMI conversions at the command line

Besides using the XMI Toolkit SmartGuide and Browser to perform XMI conversions, the XMI Toolkit includes several beans to convert between Rose, Java, UML XMI, and Java XMI.

To run from the command line, you must complete the following:

1. Set up an environment variable called IBMXMITK whose value is X:\VAJava\ide\tools\com-ibm-psh-viewer.
2. Append X:\VAJava\ide\tools\com-ibm-psh-viewer to the end of the PATH environment variable,
   where X:\VAJava is the directory in which VisualAge for Java is installed.

Important! JDK 1.2.2 must be installed on your system in order for the XMI Toolkit to run.

File suffix conventions
All XML files used by the Toolkit are XMI 1.0 standard compliant. All XMI files are also XML files, and therefore have a .XML extension. Do not use a .XMI extension, since XMI files may be processed by any XML software.

The following is a list of common file suffixes and their meanings:

- **.XML** XMI-compliant XML file
- **.DTD** XMI DTD
- **.BAT** Batch command files for Windows
- **.ZIP** Standard ZIP compression files for bundling XML
- **.MDL** Rose model files
- **.CAT** Rose logical package files

The javaMM.dtd and um11i.dtd are located in the com-ibm-psh-viewer\docs subdirectory underneath the tools directory. You should use only version 2.0.15 of the XML4J Parser.

A set of batch files is provided in the com-ibm-psh-viewer directory that you can use at the command line to easily perform conversions between the different forms of data. The table below summarizes which batch files perform which conversions.

.bat File Name	From	To

r2j.bat	Rose	Java
j2r.bat	Java	Rose
r2x.bat	Rose	UML XMI
x2r.bat	UML XMI	Rose
r2d.bat	Rose	XMI DTD

These batch files can be used to perform the following conversions:

- r2j file1 directory2 directory3 directory4 — Convert Rose model file1 to Java in directory4, writing the uml.zip file to directory2 and the java.zip file to directory3
  For example, r2j d:\mydir\mymodel.mdl d:\mydir d:\mydir\java d:\mydir\java\
- j2r directory1 directory2 directory3 file4 — Convert Java in directory1 to Rose file4, writing the java.zip file to directory2 and the uml.zip file to directory3
  For example, j2r d:\mydir\java d:\mydir\java d:\mydir d:\mydir\mymodel.mdl
- r2x file1 directory2 — Convert Rose model file1 to XMI files in directory2
  For example, r2x d:\mydir\mymodel.mdl d:\mydir\xmifiledir
- x2r directory1 file2 — Convert XMI files in directory1 to Rose file2
  For example, x2r d:\mydir\xmifiledir d:\mydir\mymodel.mdl
- r2d file1 file2 — Convert Rose file1 to XMI DTD file2
  For example, r2d d:\mydir\mymodel.mdl d:\mydir\model.dtd

Note: Property-value pairs may be included in the command stream when using r2j.bat and j2r.bat. For details, see the r2j.bat and j2r.bat files. *r2j.bat and j2r.bat can be used only on Windows NT 4.0 or 2000. You cannot use them on Windows 98.*

A file named xmi.bat is also provided which can be used to run a series of beans, one after the other, to chain together a series of XMI transformations. Specify the conversion beans for the transformation you want to perform as arguments to the xmi.bat file. Running the xmi.bat file with no arguments launches the XMI Toolkit SmartGuide. The xmi.bat file runs the Toolkit bean as follows:

```
java -mx255m -Dnobr -cp %IBMXMITK%\XMITool.jar;%CLASSPATH%
com.ibm.xmi.xmi10.Toolkit %1 %2 %3 %4 %5 %6 %7 %8 %9
```

For maximum flexibility, you may make this call directly, instead of using the batch file.

The table below summarizes the beans that you can use with the xmi.bat file.

Bean Name	**From**	**To**
R2X	Rose	UML XMI
X2R	UML XMI	Rose
J2X	Java	Java XMI
X2J	Java XMI	Java
JX2UX	Java XMI	UML XMI
UX2JX	UML XMI	Java XMI
DTDGen	Rose	XMI DTD

Command line options
The syntax for using the beans is:

```
xmi(-quiet true) Bean1 Bean2(-option1 value1 -option2 value2) Bean3
```

This example runs Bean1, then Bean2, then Bean3. Bean2 has parameters, specified using the - prefix. Options for the beans are enclosed in ( ) to keep the options visually associated with the bean they set.

To show the settings, a ″?″ may be used in place of a bean (for example, type “xmi ?”). When the Toolkit information is printed, each bean’s options are shown, along with a description, a type, a default value, and a valid range if available. To print the expert mode settings, use ″??″ in place of a bean (for example, type ″xmi ??″).

Converting from Rose to an XMI DTD, the r2d batch runs:

```
java -mx255m -cp %IBMXMITK%\XMITool.jar;%CLASSPATH%
com.ibm.xmi.xmi10.Toolkit DTDGen(-roseMdl %1 -file %2)
```

RELATED TASKS

Performing an XMI conversion

RELATED REFERENCES

R2X bean
X2R bean
J2X bean
X2J bean
JX2UX bean
UX2JX bean
DTDGen bean

Chapter 3. XMI Toolkit

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I

The sample Rational Rose model file provided with the XMI Toolkit — Account.mdl — is the beginnings of a simple banking sample that incorporates customer transactions with an Enterprise Information System (EIS), or host system.

In Part I of the sample, you will use VisualAge for Java's XMI Toolkit to convert a Rational Rose model to XMI, and then generate Java source from the XMI representation that was created during the conversion. You will then update the Java source in VisualAge for Java, and convert it back to a Rose model.

Preparing for the sample application
To walk through the sample, ensure that all prerequisites are met and that the development environment is properly set up.

Tip: To use the XMI Toolkit, you must adhere to the following when creating Rose models:

- Develop your models using only Rational Rose 98i, Enterprise Edition, or Rose 2000, Enterprise Edition.
- Develop your models with the Default Language set to Java. Before starting your model, select the **Tools > Model Properties > Edit** menu item in Rational Rose. When the Options notebook appears, click the **Notation** tab, and select **Java** as the Default Language. Click **OK** to set the default language and close the notebook.
- Do not use the frameworks provided with Rational Rose.
- Perform a Java syntax check on the model before you attempt to convert it to Java, and correct any errors and warnings that are detected. To invoke a Java syntax check in Rational Rose, select the **Tools > Java > Syntax Check** menu item. **Note:** You may bridge models that have warnings about unresolved types. You will need to use import statements in Java to resolve the types.

Attention: Only the contents of the Rose Logical view are used in the conversion to Java. In this version, the following modeling constructs are supported:

- Package
- Class
- Interface
- Attribute
- Operation
- Generalization Link
- Association Link
- Realization Link

Familiarizing yourself with the sample Account.mdl file

1. Create a directory to contain the sample Account.mdl file (and later the generated Java source).
   Tip: In this scenario, we create a directory E:\model to contain the Account.mdl file, and we create a directory E:\model\JavaFiles to contain the

generated Java source code.
Attention: When the XMI Toolkit is installed, the Account.mdl file can be found in the following directory:
X:\VAJava\IDE\TOOLS\com-ibm-psh-viewer\docs
where X:\VAJava is the directory in which VisualAge for Java is installed.

2. Copy the Account.mdl file to the directory you just created.
3. Start Rational Rose.
4. Load the Account.mdl file. The Account model has a set of packages and classes that might be used to model accounts in a banking application.
5. Examine the model to familiarize yourself with its structure, and then exit Rational Rose.

Generating Java source from a Rational Rose model
Use the XMI Toolkit to convert a model developed in Rational Rose into a Java project containing the generated Java source files. For this sample, we will use the provided Account.mdl file.

1. Start VisualAge for Java.
2. Launch the XMI Toolkit by selecting **Workspace** > **Tools** > **XMI Toolkit**. The XMI Toolkit Startup dialog box opens.
3. Select **Convert between a Rose Model and Java**, and click **OK**. The XMI Toolkit SmartGuide opens. Use the XMI Toolkit SmartGuide to specify the details of the conversion.
4. In the **Rational Rose Model** field, enter the directory path of the Account.mdl file. (In this scenario, we use the directory path E:\model\Account.mdl.)
5. In the **Java Project Directory** field, enter the directory path in which you want the generated Java source to be stored (E:\model\JavaFiles).
6. Ensure that the arrow is pointing right. You can change the arrow's direction by clicking it.
7. Ensure that **Display Conversion Mapping** is selected. Click **Next** to go to the Input Options page.
8. Click **Path Mapping**. The Path Mapping dialog opens.
   You would use the dialog to specify virtual paths for your Rose model by associating symbols with actual paths. By doing this, you can convert Rose models containing controlled units that use virtual paths. In this case, we do not need to specify anything on this page. Click **Cancel**.
9. Once you have completed the information on the Input Options page, click **Next** to go to the Output Options page.
   Tips:
   - On this page, you would specify the code generation option for association roles and class attributes. During conversion from a Rose model to Java source, association roles become fields within a Java class. If the association role has a cardinality greater than 1, then the corresponding field in the Java class will be either an array of the associated class or a Vector. Next, indicate whether you want accessor methods generated for the association roles and class attributes. For non-boolean attributes, accessor methods will be generated to get and set the generated Java field. For a boolean attribute, a method will be generated to set the field, along with a method to check the true/false value of the field. For association roles with cardinality greater than 1, accessors will be generated to get the array or Vector representing the association. Additionally, accessors will be generated to add elements to the association Vector or array, and to remove elements from it.

- For enhanced readability, the generated accessors follow a convention of upper-casing all words in the accessor name, except for the first word. Therefore, getBalance() would be generated for a field called `balance`. Because of this convention, you should begin all your UML attributes and role names with a lower-case letter. Although you can begin them with upper-case letters, if you had both a field named `balance` and another named `Balance` in the same class, each would have a generated accessor named `getBalance()`, which would cause a Java compiler error.

10. Select the **Vector** radio button, and select both the **Generate accessors (add/remove/get) for association roles** and the **Generate accessors (get/set/is) for attributes** check boxes.
11. Click **Finish** to start the Rose model to Java source conversion. A message will indicate that the conversion is complete. Click **OK** to close the message box.

The XMI Toolkit Browser is updated, displaying two tree structures. The tree structure on the left side displays the UML XMI representation of the Rose model that was used as input. The tree structure on the right displays the Java XMI representation of the Java source that was generated during the conversion.

Familiarizing yourself with the UML to Java mapping in the XMI Toolkit Browser

Before working with the generated Java files, examine the UML to Java mapping in the XMI Toolkit browser. The mapping is shown in the two tree structure views.

1. Click any node in one tree structure to highlight the corresponding node in the other tree structure.
2. Expand and collapse the nodes, and click the inner nodes to see how the Rose model has been mapped into Java.

Modifying the generated Java source

In this section, you will import the generated Java code into the IDE, and then modify it.

1. In the VisualAge for Java IDE, select **File** > **Import**. The Import SmartGuide opens.
2. Select **Directory** as the import source, and click **Next**.
3. In the Directory field, enter the full directory path containing the Java source (E:\model\JavaFiles).
4. Select to import only the **.java** files.
5. Type `Account` for the project name.
6. Click **Finish**. The Account project now appears in the Workbench.
7. In the Account project, make the following changes:
   a. Add a new package named `com.eplace.ecustomer`.
   b. In the com.eplace.ecustomer package, add a new class called `CustomerInformation`. This class will be used to hold information about an ebank customer.
   c. Add the following fields of type `String` to the CustomerInformation class: `name`, `address`, and `pin`.
   d. In the Account class, add a field called `accountHolder` of type `CustomerInformation`.
   e. Add a second field called `active` of type `boolean`.
8. Now, export the modified source to the directory path in which the generated Java source is stored (E:\model\JavaFiles):
   a. In the Workbench, left-click the Account project to select it.

b. Select **File** > **Export**. The Export SmartGuide opens.

c. Export the **.java** and **.class** files of the Account project to the E:\model\JavaFiles directory. When prompted, select to overwrite the old versions.

Generating a new Rational Rose model from the modified Java source
Use the XMI Toolkit to generate a new Rational Rose model from the updated Java source.

Important! Keep a backup copy of your Rose model (i.e. the .mdl and .cat files).

Tip: To convert Java source to a Rational Rose model, you must ensure that the following conditions are met:

- All source files in your Java project must be successfully compiled.
- Both the Java source files and compiled class files must reside on the file system.

Before generating a new Rational Rose model from the Java source, you must first export the Java source and class files from VisualAge for Java into the project directory you used before.

Use the XMI Toolkit SmartGuide to specify the details of the conversion from Java source to Rose model.

1. In the XMI Toolkit Browser, select **File > Perform a Conversion**. The XMI Toolkit SmartGuide opens.
2. In the **Java Project Directory** field, enter the full directory path of the Account Java project directory (in our scenario, this is E:\model\JavaFiles).
3. In the **Rational Rose Model** field, enter the full directory path of the Account.mdl file (E:\model\Account.mdl).
4. Ensure that the arrow is pointing left.
5. Ensure that **Display Conversion Mapping** is checked. Click **Next** to go to the Input Options page.
6. The Input Options page lists all the packages that the XMI Toolkit has found in your Java project. Click **Select All** to select both the com.eplace.ebank and com.eplace.ecustomer packages.
7. Click **Next** to go to the Output Options page.
8. The Output Options page allows you to specify virtual path mappings for your generated Rose model. Click **Path Mapping**. The Path Mapping dialog box opens. You can use the Path Mapping dialog box to associate symbols with actual paths, as you would do in Rational Rose. Since the Account model has no controlled units, click **Cancel** to close the dialog box.
9. Click **Finish**. This starts the XMI conversion from the Java source to the Rose model. When the conversion completes, click **OK** to clear the completion message.
10. The XMI Toolkit Browser now displays the Java XMI representation of the Java project in the tree on the left, and the UML XMI representation of the updated Rose model on the right. As before, selecting a node in one tree will highlight the corresponding node in the other tree. In the XMI Toolkit Browser, select the CustomerInformation class that you added to the com.eplace.ecustomer package. (The node representing this class is labeled `com.eplace.ecustomer.CustomerInformation`.) Note that the UML XMI tree expands. The class you added is highlighted in the UML tree, appropriately nested in the ecustomer package that has been added to the Rational Rose model. Select other nodes in the two tree structures to examine the Java-to-UML mapping.

Viewing the UML-UML differences
Now, you can view the differences between the old and new versions of the UML XMI files.

1. The new version of the Rose model contains the original UML constructs, as well as changes that were made in the Account project using VisualAge for Java. To see how the changes have been incorporated into the Rose model, you must compare the new UML XMI representation with the previous one:
   a. Select **File** > **Show UML-UML Differences** in the XMI Toolkit Browser. The Select a UML XMI zip File dialog box opens.
   b. Select the **uml.zip** file. The uml.zip file contains the most recent UML XMI representation of your Rose model. The XMI Toolkit will compare this representation with the previous version contained in uml_old.zip. (**Note:** The file uml_old.zip is a copy of the previous uml.zip.)
   c. To view the differences, click **Open**.
2. The differences between the old and new versions of the UML XMI files are displayed in tabular form in the lower portion of the XMI Toolkit Browser. The tree structure on the left displays the new version of the Rose model, while the tree on the right displays the old version.
   In the Differences pane, select an added element to highlight the appropriate node in the tree structure on the left.

In "Sample: Roundtripping between Rational Rose and VisualAge for Java - Part II," you will do the following:

1. Examine the newly generated Rose model to see how the changes made in Java have been incorporated into the original model.
2. Make updates to the new version of the model.
3. Convert the new model to Java.
4. Compare the new version of the Java XMI with the old version of the Java XMI.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part II

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part II

In Part II of the sample, you will use Rational Rose to modify the updated Rose model, and then use the XMI Toolkit to convert the modified Rose model to Java source.

Modifying the updated Rational Rose model
In this section, you will modify the updated model in Rational Rose.

1. Launch Rational Rose and load the new version of the Account model. When the model has loaded, verify that the constructs you added in VisualAge for Java have been included in the model.
2. When you open the eplace package in the Class Diagram pane, you will notice that the ebank package is visible, but the ecustomer package that you earlier added to the Java source code does not appear. Do the following:
   a. In the tree structure view of the Account model, expand the **eplace** package.

b. Drag and drop the **ecustomer** package onto the Class Diagram pane for the eplace package, so that the ecustomer package is now visible in this pane. (This associates user interface information with the ecustomer package, so that the next time you load the model, the ecustomer package will appear in the Class Diagram pane for the eplace package.)
c. Double-click the **ecustomer** package in the Class Diagram pane to open it.
d. Drag and drop the **CustomerInformation** class from the tree structure view onto the Class Diagram pane for the ecustomer package. (This associates user interface information with the CustomerInformation class.)

Tip: In Rational Rose, the Class Diagram pane does not immediately reflect the updated model. To recreate the diagram to reflect the updated model, you must always drag the package, class, or interface from the tree structure view to the Class Diagram pane.

3. In Rational Rose, make the following changes to the Account.mdl file:
   a. Open the Class Diagram for the ebank package. Use the Unidirectional Association tool to draw an association from the Account class to the Branch class. Update the association using the Standard Specification, according to the following:
      - Association Name: home
      - RoleA: homeBranch, navigable, cardinality 1..1
      - RoleB: homeAccount, navigable, cardinality 1..n
   b. In VisualAge for Java, you had added an attribute named active to the Account class. This attribute serves the same purpose as the accountStatus attribute in the Checking class. It also serves the same purpose as the active attribute in the Savings class. Since both of these classes inherit from the Account class, you can remove these two local attributes, and use the attribute that is inherited from the Account class:
      1) Delete the **accountStatus** attribute and its accessors from the Checking class.
      2) Delete the **active** attribute and its accessors from the Savings class.
4. Save the Rose model, and exit Rational Rose.

Converting the modified Rose model to Java source
Now, you will convert the modified Rose model to Java source. You will then view the UML to Java conversion mapping in the XMI Toolkit.

1. Launch the XMI Toolkit if it is not already launched. When the XMI Toolkit Startup dialog box opens, select **Convert between a Rose model and Java**, and click **OK**. The XMI Toolkit SmartGuide opens. (If the XMI Toolkit was already launched, then select **File** > **Perform a Conversion** in the XMI Toolkit. The XMI Toolkit SmartGuide will open.)
2. **Important!** For both the **Rational Rose Model** field and the **Java Project Directory** field, ensure that the same paths are used as in the section “Generating Java source from a Rational Rose model” in Part I of this sample.
3. Ensure that the arrow is pointing right.
4. Click **Next** to go to the Input Options page of the XMI Toolkit SmartGuide. Accept the defaults, and click **Next**. The Output Options page of the XMI Toolkit SmartGuide opens. Accept the defaults, and click **Finish**.
   Note: The last version of the Java source and .class files are moved to a backup directory (E:\model\JavaFiles_old) before the conversion starts.
5. A message confirms that the Rose model to Java source conversion has completed. Click **OK** to clear the message. The Java source files for the

updated Account model will be written into a directory hierarchy contained beneath the project directory according to the package structure.

6. As with the previous conversions, the XMI Toolkit will automatically display the UML XMI and Java XMI information that was created during the conversion. Examine the UML-to-Java mapping in the XMI Toolkit browser by clicking the nodes in the two tree structures.

Viewing the Java-Java differences
You can view the differences between the old and new versions of the Java XMI files.

1. Since the new version of the Java project incorporates the changes you made to the Rose model, you can compare the new version with the previous version. To see how the changes have been incorporated into the Java project, you must compare the new Java XMI representation with the previous one:
   a. Select **File** > **Show Java-Java Differences** in the XMI Toolkit browser. The Select a Java XMI zip File dialog box opens.
   b. Select the **java.zip** file. The java.zip file contains the most recent Java XMI representation of the Account Java project. The XMI Toolkit will compare this representation with the previous version contained in java_old.zip. (**Note:** The file java_old.zip is a copy of the previous java.zip.)
   c. To view the differences, click **Open**.
2. The differences between the old and new versions of the Java XMI files are displayed in tabular form in the lower portion of the XMI Toolkit Browser. The tree structure on the left displays the new version of the Java XMI files, while the tree on the right displays the old version. In the Differences pane, select an added element to highlight the appropriate node in the tree structure on the left. Select a deleted element to highlight the appropriate node in the tree structure on the right.

You have completed the roundtripping between Rational Rose and VisualAge for Java. Using VisualAge for Java's XMI Toolkit, you were able to convert between a Rose model and Java source, to examine Java/UML conversion mappings, and to view differences between old and new versions of the project.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Sample: Roundtripping between Rational Rose and VisualAge for Java - Part I

UML/Java mappings

UML to Java mapping

The following table summarizes the mapping that the XMI Toolkit uses when it converts a Rose (UML) model to Java.

Rose Construct	Java Construct	Notes
Model	None	
Package	Package statement	Nested packages as dot-separated elements in the package statement
Class	Class	

Class with ″primitive″ stereotype	Class	Case must match on ″primitive″ spelling
Class with ″structure″ stereotype	Class	Case must match on ″structure″ spelling
Class with ″enumeration″ stereotype	Class	Case must match on ″enumeration″ spelling
Interface (or Class with ″Interface″ stereotype)	Interface	Case must match on ″Interface″ spelling
Operation	Method	
Parameter	Parameter	
Attribute	Field	Accessor methods optionally generated
Generalization, Class to Class	Extends clause	
Generalization, Interface to Interface	Extends clause	
Realization, Class to Interface	Implements clause	
Association	None	
Association Role	Field	Vector or array if cardinality > 1, accessor methods optionally generated. **Attention:** Fields corresponding to association roles should not be deleted or modified (including changing their types) in the generated Java source files. Changes to association roles should be made only in your Rose model.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing an XMI conversion
Going from a Rose model to Java
Going from Java to a Rose model
Performing XMI conversions at the command line

Java to UML mapping

The following table summarizes the mapping that the XMI Toolkit uses when it converts Java to a Rose (UML) model.

Java	Rose	Notes
Package statement	Package	Nested packages for each dot-separated element
Class	Class	
Interface	Class—Interface stereotype	
Method	Operation	
Parameter	Parameter	
Field	Attribute	Association role, if from a previous association in Rose
Extends clause, class to class	Generalization	

Java	Rose	Notes
Extends clause, interface to interface	Generalization	
Implements clause, class to interface	Realization	

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing an XMI conversion
Going from a Rose model to Java
Going from Java to a Rose model
Performing XMI conversions at the command line

XMI Toolkit beans

Toolkit bean (xmi)

The Toolkit bean, which is invoked by the xmi.bat file, runs a series of beans.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
`-debug*`	`Print debug information.`	`(boolean, false)`
`-quiet`	`Suppress toolkit output.`	`(boolean, false)`

For example, xmi(-quiet true) Bean1 Bean2(-option1 value1 -option2 value2) Bean3.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

R2X bean
X2R bean
J2X bean
X2J bean
JX2UX bean
UX2JX bean
DTDGen bean

DTDGen bean

DTDGen uses the Reference implementation portion of the Toolkit to create DTDs by strictly following the algorithms in the XMI specification.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
-file	.dtd file to write.	(String)
-version	Specify XMI version for the DTD (either 1.0 or 1.1)	(String, 1.1)
-roseMdl	Rose model	(String)

For example, xmi DTDGen(-roseMdl D:\model.mdl -file D:\model.dtd).

RELATED TASKS
Performing XMI conversions at the command line

RELATED REFERENCES
Toolkit bean (xmi)
R2X bean
X2R bean
J2X bean
X2J bean
JX2UX bean
UX2JX bean

J2X bean

J2X reads Java files and generates Java XMI files.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
-sources*	Location of source files.	(String)
-packageList	List of package names.	(String, null)
-rootPath	Root package directory.	(String, .)
-outPath	Output directory or destination zipfile.	(String, java.zip)
-doMerge*	Run diffmerge on each class.	(boolean, false)

Notes on options for J2X
The rootPath option specifies the root directory of the package hierarchy for your compiled Java .class files. If the source files reside in the same directories as the corresponding .class files, you should accept the default value for the sources option, which is an empty string. If the source files reside in a directory structure mirroring that of the .class files but under a different root directory, you should supply that directory to the sources option and include an asterisk at the end of the directory name. Because of the asterisk, you must quote the argument to the

sources option. The outPath option allows you to write either a single zip file or individual XMI (.xml) files under the specified directory, for example:

```
xmi J2X(-rootPath d:\j\bin -sources ″d:\j\src*″ -outPath d:\j\java.zip)
```

Use the packageList option if you want to specify a subset of the packages that are contained beneath the directory you provide to the rootPath option. The value of the argument should be a quoted, blank-separated list of the packages you want to convert from Java to Java XMI. The default value (null) indicates to include all the packages beneath the rootPath directory, for example:

```
xmi J2X(-rootPath d:\j -outPath d:\j\java.zip -packageList ″com.a.b com.c.d″)
```

A process called diffmerge is run if the doMerge option is set to true. diffMerge performs differencing and merging of the current XMI information with the XMI information from the last conversion. If you are roundtripping between Java and Java XMI, set the doMerge flag to true so that your changes in Java are differenced and merged with the previous version saved in Java XMI. If you are only going from Java to Java XMI, you can set doMerge to false (the default). Also, if you are using J2X as the first bean in a sequence to go from Java to Rose, you will also want to set doMerge to true if you will be doing roundtripping between Rose and Java.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

Toolkit bean (xmi)
R2X bean
X2R bean
X2J bean
JX2UX bean
UX2JX bean
DTDGen bean

JX2UX bean

JX2UX reads a Java XMI zip file and saves to a UML XMI zip file.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
-umlZip	Output zip file for uml model.	(String, uml.zip)
-roseMdl*	Rose output info.	(String)
-setDTD*	Save xml file with dtd.	(boolean, false)
-javaZip	Input zip file with java model.	(String, java.zip)

For example, xmi JX2UX(-javaZip d:\java.zip -umlZip d:\uml.zip -roseMdl d:\model.mdl)

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

Toolkit bean (xmi)
R2X bean
X2R bean
J2X bean
X2J bean
UX2JX bean
DTDGen bean

R2X bean

R2X and X2R are batch conversion functions to be used when you have a huge model which won't all fit in memory. Only a part of the model is in memory at any time. R2X and X2R map multiple MDL/CAT files one-to-one with XMI files. Zip access and directory path configuration are also available.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
-extensions*	Create extensions.	(boolean, false)
-pathMap	Path map for .cat files.	(String)
-file	.mdl file to load.	(String)
-core	Core information only.	(boolean, true)
-xmiZip*	Zip file to save XMI files.	(String)
-noPackage*	Do not load packages.	(boolean, false)
-setDTD*	Save xml file with dtd.	(boolean, false)
-notice*	File name contains your notice.	(String)
-doMerge*	Run diffMerge against old XMI files or zipfile.	(boolean, false)
-xmiDir	Directory to save XMI files.	(String, .)
-ui*	Preserve ui information.	(boolean, true)

Notes on options for R2X
Syntax for pathMap: “var1=value1,var2=value2...” Note that the pathMap is NOT case-sensitive. The quotes around the pathMap argument are required.

You can use the notice option to include information from a file (for example, a copyright) into every generated XMI file. Provide the complete file name as the argument to the notice option.

For example, xmi R2X(-file d:\myModel.mdl -xmiDir d:\XMIFiledir -notice d:\mydir\copyright.txt)

Extensions include data needed for round-tripping (for example, going back and forth between Rose and UML XMI). UI information includes data for the layout of your Rose model in the Class Diagram in the Rose Logical View. Setting core to true excludes extensions and ui information.

A process called diffmerge is run if the doMerge option is set to true. diffMerge performs differencing and merging of the current XMI information with the XMI information from the last conversion. If you are roundtripping between Rose and UML XMI, set the doMerge flag to true so that your changes in Rose are differenced and merged with the previous version saved in UML XMI. If you are only going from Rose to UML XMI, you can set doMerge to false (the default).

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

Toolkit bean (xmi)
X2R bean
J2X bean
X2J bean
JX2UX bean
UX2JX bean
DTDGen bean

UX2JX bean

UX2JX reads a UML XMI zip file and saves to a Java XMI zip file.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
-mapAsArray*	Convert association roles to array not vector.	(boolean, false)
-umlZip	Input zip file with uml model	(String, uml.zip)
-setDTD*	Save xml file with dtd.	(boolean, false)
-javaZip	Output zip file for java model.	(String, java.zip)

For example, xmi UX2JX(-umlZip d:\drv\uml.zip -javaZip d:\drv\java.zip)

RELATED CONCEPTS
XMI Toolkit - overview

RELATED TASKS
Performing XMI conversions at the command line

RELATED REFERENCES
Toolkit bean (xmi)
R2X bean
X2R bean
J2X bean
X2J bean
JX2UX bean
DTDGen bean

X2J bean

X2J reads Java XMI files and generates Java.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
`-assocAcc*`	`Generate accessors for association roles.`	`(boolean, false)`
`-outPath`	`Output directory for .java files.`	`(String, .)`
`-attribAcc*`	`Generate accessors for attributes.`	`(boolean, false)`
`-imports`	`List of imports (;-separated) to merge into code.`	`(String, null)`
`-xmiList`	`List of xml or zip filenames.`	`(String, java.zip)`

For example, xmi X2J(-xmiList d:\drv\uml.zip -outPath d:\myJavaProject)

RELATED CONCEPTS
XMI Toolkit - overview

RELATED TASKS
Performing XMI conversions at the command line

RELATED REFERENCES
Toolkit bean (xmi)
R2X bean
X2R bean
J2X bean
X2J bean

JX2UX bean
UX2JX bean
DTDGen bean

X2R bean

R2X and X2R are batch conversion functions to be used when you have a huge model which won't all fit in memory. Only a part of the model is in memory at any time. R2X and X2R map multiple MDL/CAT files one-to-one with XMI files. Zip access and directory path configuration are also available.

Bean options
In parentheses is the type of each option and the default value if applicable. Expert mode options are denoted with an *.

Bean option	Description	Type, Default value
`-pathMap`	`Path map for .cat files.`	`(String)`
`-file`	`.mdl file to write.`	`(String)`
`-umlPath`	`Directory to load XMI files or zip file path name.`	`(String)`
`-setDTD*`	`Resave xml file with dtd.`	`(boolean, true)`

For example, xmi X2R(-umlPath d:\uml.zip -file d:\model.mdl)

Notes on options for X2R
Syntax for pathMap: "var1=value1,var2=value2..." Note that the pathMap is NOT case-sensitive. The quotes around the pathMap argument are required.

RELATED CONCEPTS

XMI Toolkit - overview

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

Toolkit bean (xmi)
R2X bean
J2X bean
X2J bean
JX2UX bean
UX2JX bean
DTDGen bean

DTD generation conventions

There are two options that may be used to indicate that a UML class should correspond to an XMI string or enumeration. Classes marked with one of the two special stereotypes, <<primitive>> or <<enumeration>>, will not appear in the DTD, but instead provide places for XMI documents to contain data values.

If a class C has a stereotype of <<primitive>>, attributes of type C will be marked as PCDATA in the generated DTD. If a class E has stereotype <<enumeration>>, an attribute of type E will be marked as an enumeration using the xmi.value attribute, where the choices allowed in the DTD will correspond to the names of the attributes in class E.

For example, if a class called StringValue with stereotype <<primitive>> was defined in a model, and another class Person had an attribute called Name of type StringValue, then the DTD would contain <!ELEMENT Person.Name(#PCDATA)> instead of <!ELEMENT Person.Name (StringValue)>.

RELATED TASKS

Generating a DTD

RELATED REFERENCES

DTDGen bean

XMI file information

The contents of the XMI files follow the XMI standard for UML 1.1 models. The DTD UML11i.DTD is the same DTD as in the XMI specification with the relaxed multiplicity rules option.

To support information beyond that directly specified in UML 1.1, XMI extensions are used. For example, UML Notation information is stored in UMLNotation extensions as it is not yet part of the official UML metamodel, but still very useful information for interchange.

The Toolkit uses the convention IXT (IBM XMI Toolkit) to store extension information as sets of tag-value pairs. The internal DTD at the beginning of the XMI documents defines the format of the extensions used in the Toolkit.

Some information that is needed for round-trip exchange with some tools is preserved in the extensions. For example, Rose-specific object identifiers, called QUIDs, are preserved so that files can be round-tripped back to the original form, preserving their ″native″ IDs. In the future, tools should accept UUIDs instead of requiring a proprietary format.

UUIDs
The file Guid.dll, which resides in the Toolkit installation directory, contains an executable compiled C++ program that calls Window’s CoCreateGuid function to generate a GUID and convert it to the DCE format of a UUID. If the Guid.dll is not used, DCE-lite UUIDs are generated by the Java in the Toolkit. Therefore, if you prefer to use DCE-lite UUIDs, you should remove the Guid.dll file from the Toolkit installation directory (com-ibm-psh-viewer).

ZIP files and XMI
XMI supports ZIP files. The .zip format is the standard format available through many utilities. You can zip or unzip using these utilities in conjunction with the Toolkit.

Results appear to be about 95 percent or better compression for a speed reduction of less than 20 percent while loading and no reduction while saving.

RELATED TASKS

Performing XMI conversions at the command line

RELATED REFERENCES

Toolkit bean (xmi)
R2X bean
X2R bean
J2X bean
X2J bean
JX2UX bean
UX2JX bean

Notices

Note to U.S. Government Users Restricted Rights — Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this document in other 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.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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 OR CONDITIONS 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 the 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.

Programming interface information

Programming interface information is intended to help you create application software using this program.

General-use programming interfaces allow the customer 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