IBM Rational Developer for System z

Advanced COBOL Development Technical Preview

Version 7.5 Technical Preview

IBM Rational Developer for System z

Advanced COBOL Development Technical Preview

Version 7.5 Technical Preview

Note:
Before using this information and the product it supports, read the information in "Notices," on page 43.

First Edition March 2008

Contents

Chapter 1. Introduction to Processes and Components

This is the Advanced COBOL Development technical preview.

The UML to COBOL Process

This technology preview is intended to help bridge the UML model-driven architecture process to the COBOL development process, by allowing the modeling of data structures and programs and then, the generation of COBOL source code using these models.

The inclusive modeling-to-development process uses both the Rational® Software Architect (RSA) and the Rational Developer for System z™ (RDz) platforms. Additionally, an RSA user may also export an Eclipse Modeling Framework (EMF) representation of their assembled model, which can then be imported by an RDz user, and the COBOL generated within RDz.

The modeling process requires several models and transformations to complete the full transformation from UML to COBOL. Initial modeling of data structures and programs is accomplished in RSA by using language-neutral UML profiles. The initial model is then transformed into a second model, this one using COBOL language-specific profiles. This generated COBOL model can be customized further, and its programs detailed using activity diagrams. The next step of the transformation produces two outputs:

- COBOL source code, if the modeler has RDz installed along with RSA.
- A file containing an EMF representation of the modeled data structures and code, in an intermediate state of transformation.

This EMF representation can be shared with an RDz user, where the final transformation into COBOL source code can be completed.

The Advanced COBOL Editor

The Advanced COBOL Development technology preview provides features and capabilities to enhance the COBOL editing experience. These features include:

1. Real-time syntax checking with limited semantic checking; for example, unresolved references.
2. An "Open Declaration" action that allows navigation from a reference to a data item, section, or paragraph to its corresponding declaration.
3. An "Open Perform Hierarchy" action that presents a view for navigating a perform hierarchy for a given paragraph or section.
4. A "Rename" refactoring that operates on data items, paragraphs, and sections.
5. A "Remove Noise Words" refactoring that removes unnecessary noise words from the source.

For further information and available features, see Chapter 3, “Advanced COBOL Editor Features,” on page 35.

The VSAM/QSAM Wizard

The Advanced COBOL Development technology preview provides new features and capabilities to create new COBOL programs that perform I/O to VSAM and/or QSAM files via the new VSAM/QSAM Wizard.

For further information and available features, see Chapter 4, "The VSAM/QSAM Wizard," on page 41.

Chapter 2. The UML to COBOL Process

The UML to COBOL process requires the modeling of data structures and programs. The next step in the process is the generation of the COBOL source code using the generated models. This process uses both RSA and RDz as platforms. RSA is required. See the RSA documentation for further information.

Developing a model using RSA

The following sections describe how to develop a model using Rational Software Architect.

Modeling the language-neutral layer

Using the provided primitive types and stereotypes for class attributes

The solution provides a model library that contains business oriented primitive types. These primitive types augment the UML primitive types that RSA provides. Their usage is not mandatory for the rest of the tasks, but they provide a better semantic for the attributes you want to describe. The transformations to the language related layers take advantage of this extra semantic.

When you create an UML model, you can specify which model libraries you want to use. To specify which libraries to add, do the following:

1. Open your UML model (.emx file) and, in the editor view, switch to the Details tab.
2. Select **Add** under **Model Libraries**. The **Import Model Library** window opens. See Figure 1 on page 4.
3. In the **Import Model Library** window, select the **Deployed Library** radio button.
4. From the **Deployed Library** drop-down menu, select **Data Types**.
5. Select **Ok**.

Figure 1 on page 4 shows how to include the provided model library.

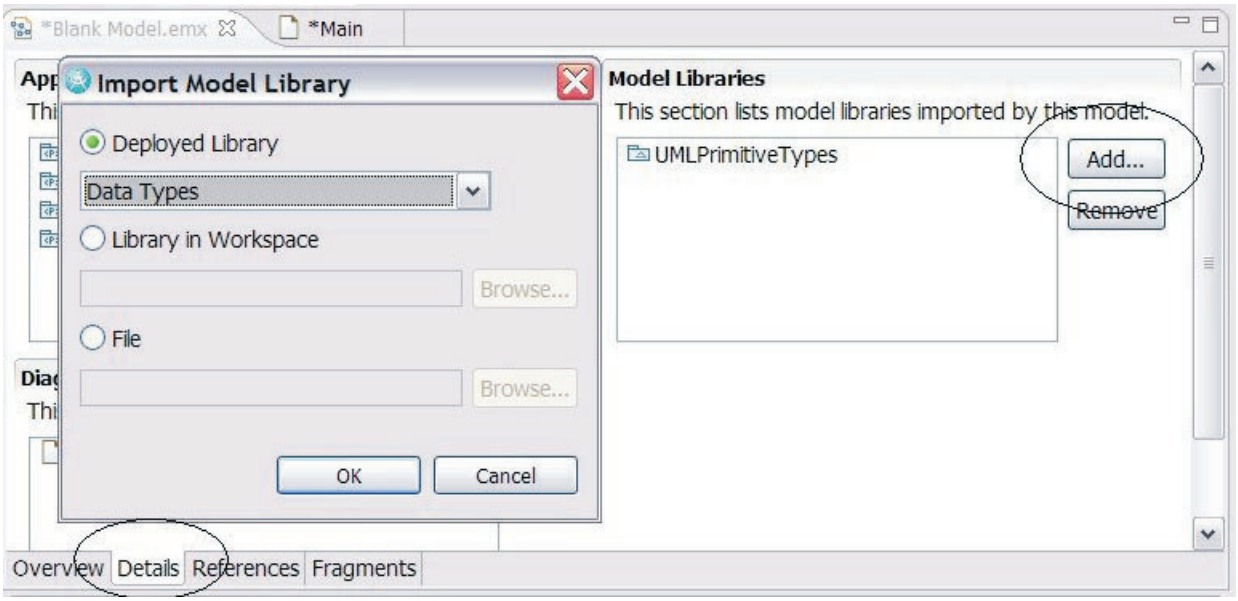

Figure 1. Including a provided model library

The model library provides the following types for attributes of a class.

- Data
- Binary
- Currency
- Date
- Float
- Integer
- String

Figure 2 on page 5 shows the types for class attributes from the **Select Element for Type** window.

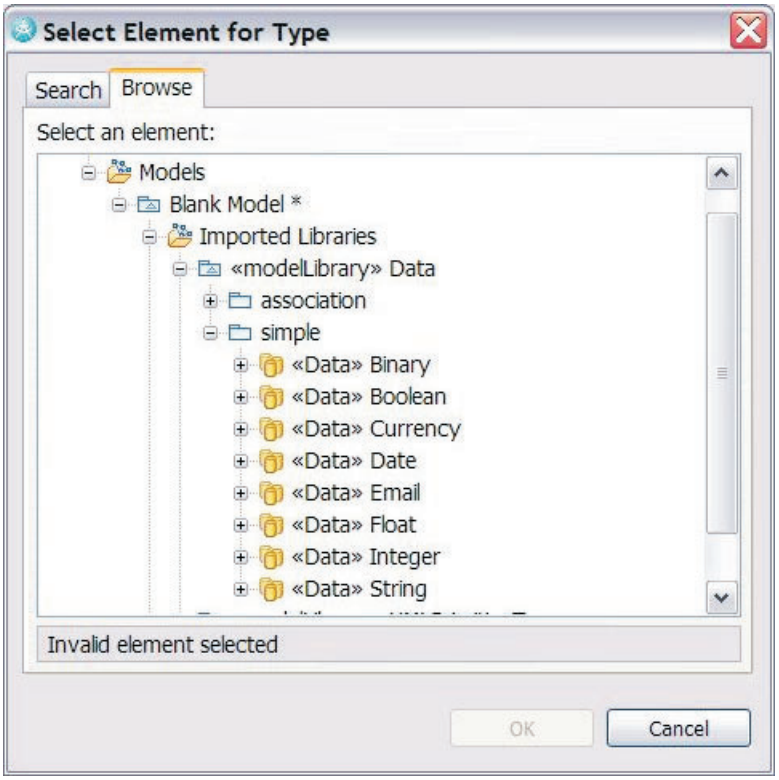

Figure 2. Types for class attributes

Include your own primitive types in the process: Most people define their own sets of primitive types in order to define basic business elementary types. Most often these types are similar in concept to the ones we provide. Since our process takes advantage of the primitive types provided by the solution, you need to customize the solution to take into account your primitive type instead of the provided ones and specify which are the appropriate stereotypes for the new primitive types. Or you can add a transformation in order to obtain a model that uses the primitive types of the solution.

Specify type properties for the attribute: The attribute primitive types provide semantics about type information. You can optionally set properties related to these types by applying a dedicated stereotype to the attribute of the class you describe in UML. These properties better refine the way transformation occurs. These stereotypes are contained within the Data profile.

To use the Data profile in your UML model, do the following steps:

1. Under **Applied Profiles** select **Add**. The **Select Profile** window opens.
2. In the **Select Profile** window, toggle **Deployed Profile**.
3. From the **Deployed Profile** drop-down menu, select **Data Profile**. See Figure 3 on page 6.
4. Select **Ok**.
5. The data profile is added to the list of profiles to apply to the model.

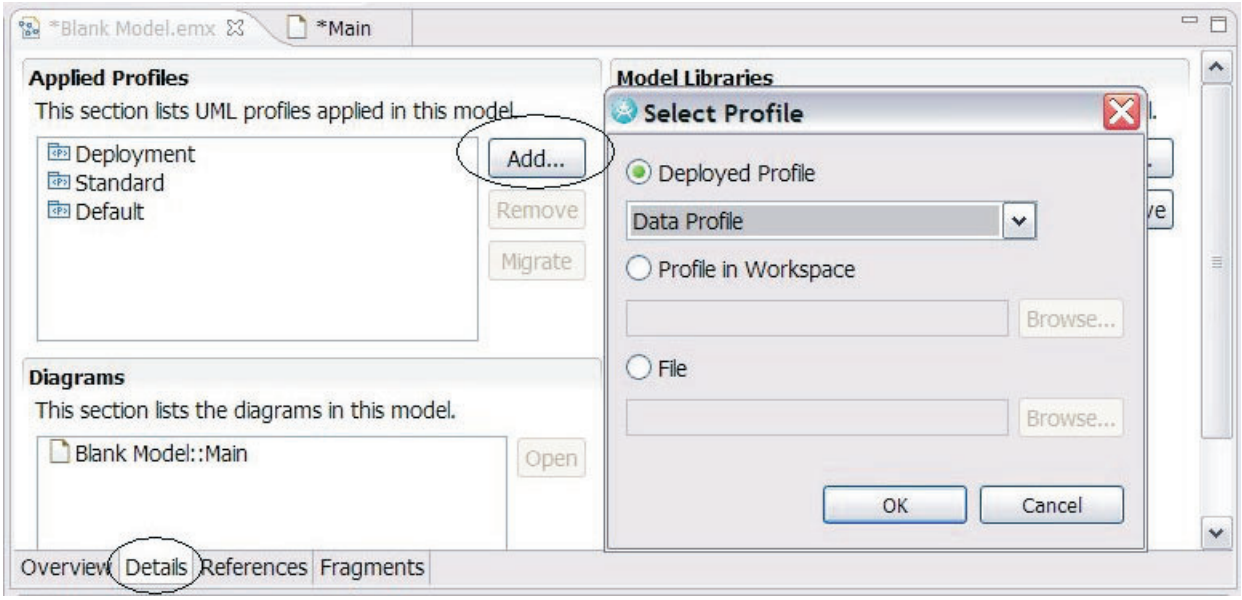

Figure 3. Adding a data profile

You can also use the stereotypes with the UML default primitive types provided by RSA. In this case, the transformation ensures that the stereotype chosen is consistent with the primitive type; that is, it ensures that a string stereotype is not applied on an integer primitive type.

Figure 4 shows the list of stereotypes that are applicable to the attributes of a class.

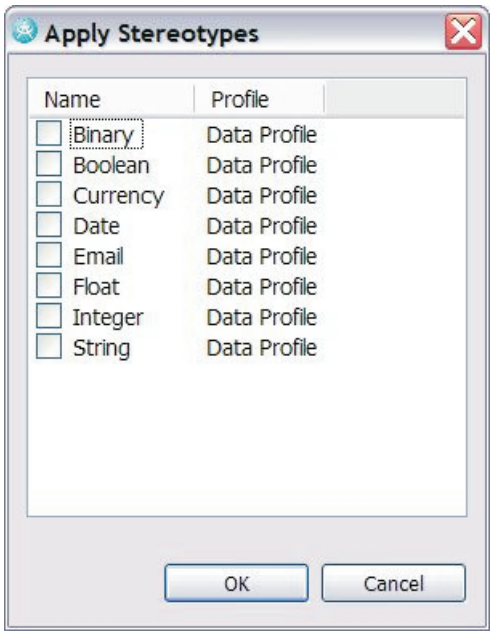

Figure 4. Applicable stereotypes

The list includes the following:

- Binary
- Boolean
- Currency

- Date
- Email
- Float
- Integer
- String

The stereotype contains properties that are language independent. These properties define the characteristics of the field more accurately. To apply such a stereotype, do the following:

1. Select the class attribute you want to apply the stereotype to and go to the stereotype pane in the Property view. Figure 5 shows the Property window.

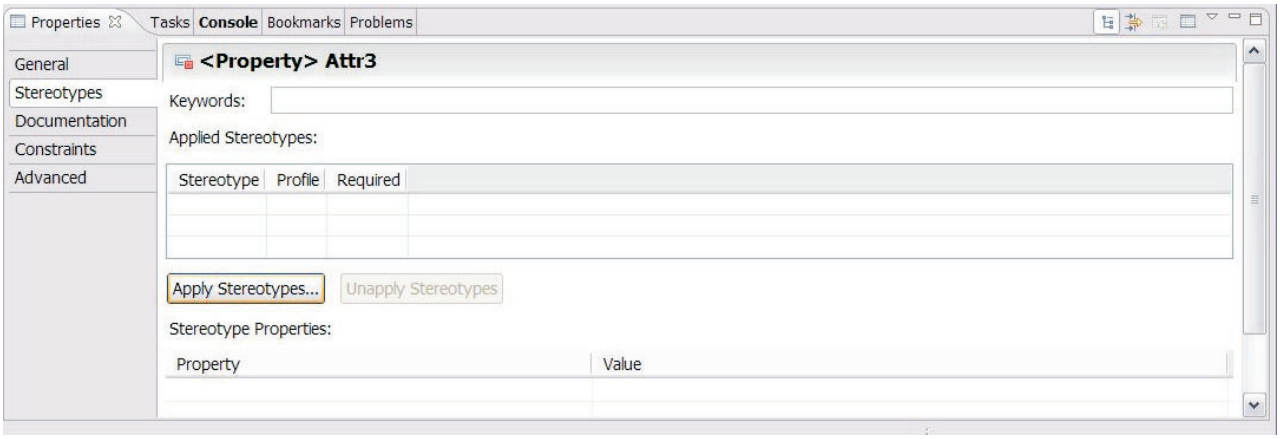

Figure 5. Properties window

2. Select the **Apply Stereotype** button. The **Apply Stereotype** window opens with the list of possible stereotypes. In Figure 6 on page 8, the String stereotype has been selected.

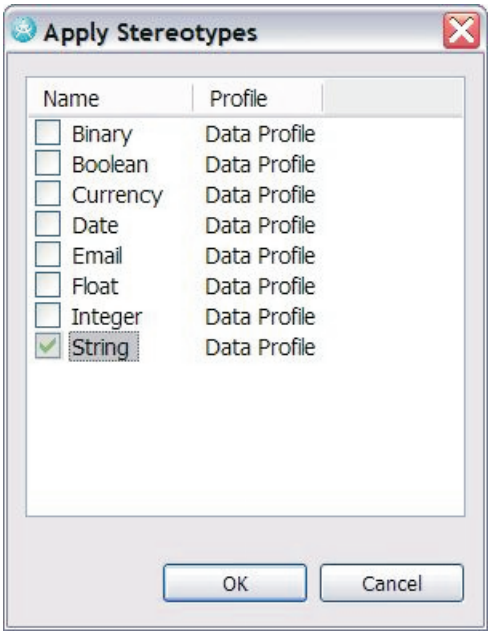

Figure 6. String is selected

Figure 7 shows that the String stereotype is now applied. You now have access to the stereotype properties that you can set. As shown in Figure 7, we have set the maxLength property.

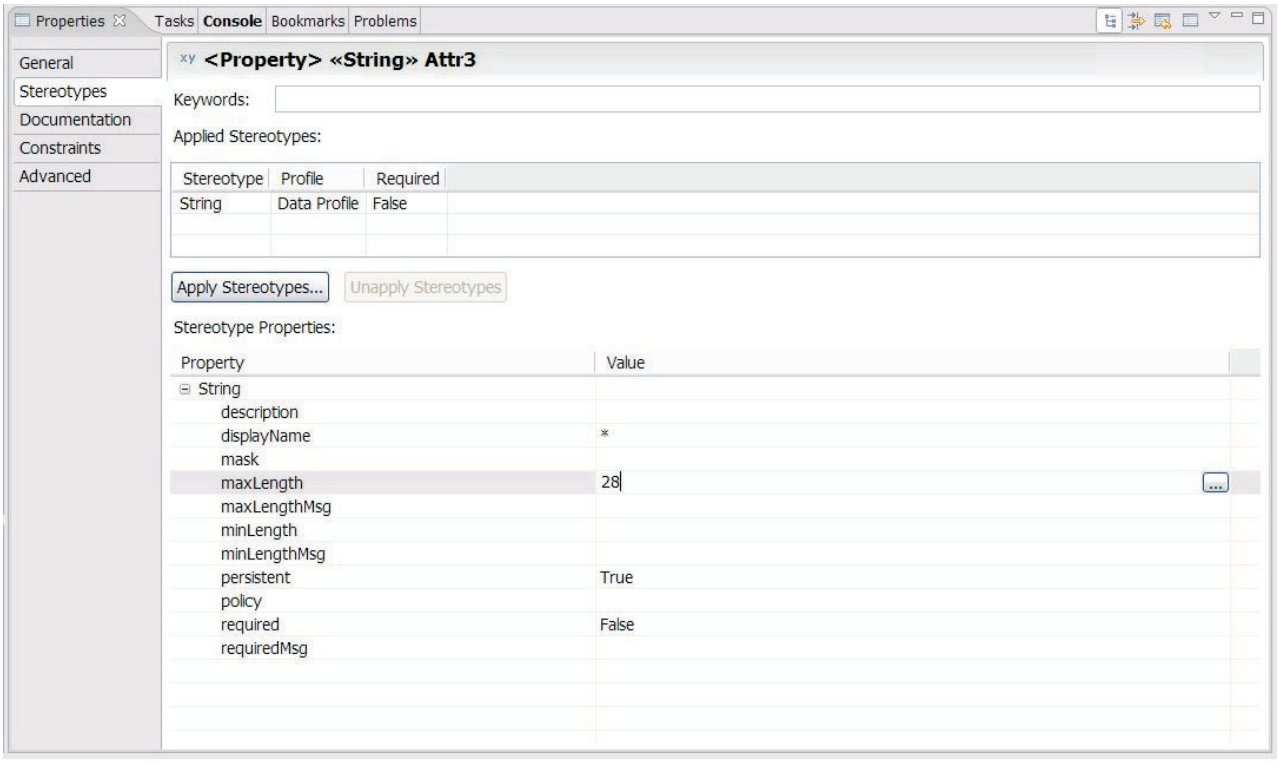

Figure 7. Stereotype properties of String

Identify the Data Object and Services (Optional): This is a recommended but optional step. The Data Profile contains a Data Object Stereotype that identifies a class as a provider of Data definitions, through its attributes and relationships.

You can also use the Service Profile in your model in order to identify the classes that provide operations (business-level operations) in your system. Identifying the service class now or in the later stage is used.

Figure 8 shows a simple model that uses the primitive types provided. This simple model also shows stereotypes for attributes and classes.
.

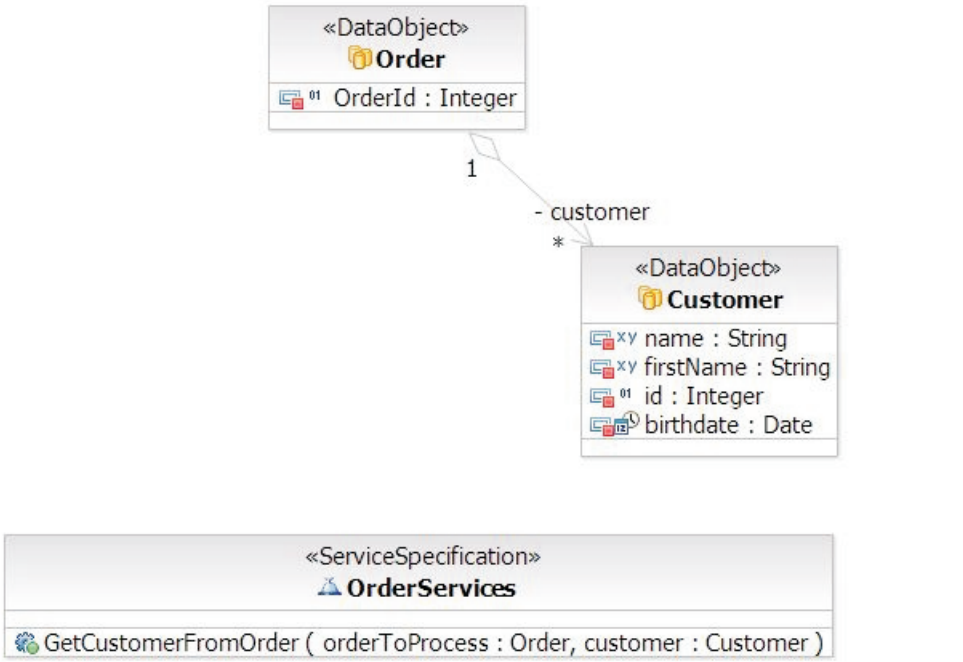

Figure 8. Simple model applying the data profile

Creating the COBOL-dependent layer

Defining the launcher for the transformation: The first time you run a transformation you must create a configuration where you will specify parameters of the transformation and that you will therefore use to launch the transformation. Figure 9 on page 10 shows how to create a transformation configuration. Do the following:

1. In the modeling window, select **File > New > Transformation Configuration**.

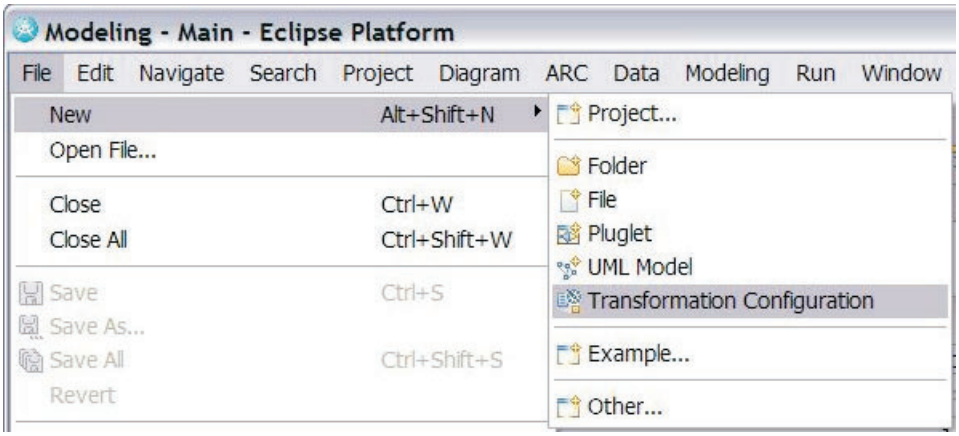

Figure 9. Creating a transformation configuration

2. The **New Transformation Configuration** window opens. In this window, you can select the transformation. Figure 10 shows the transformation configuration window for selecting a COBOL transformation.

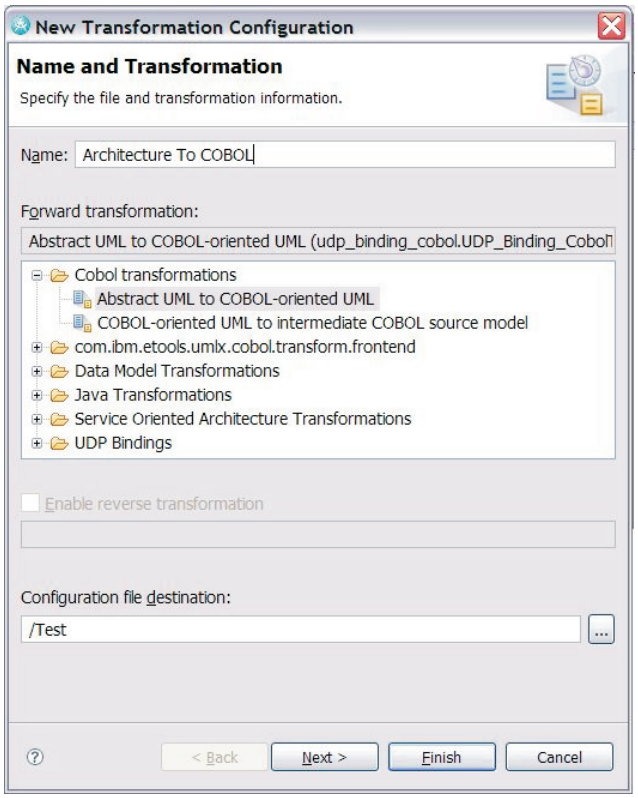

Figure 10. Selecting a COBOL transform

3. Under **Forward transformation**, highlight **Abstract UML to COBOL-oriented UML**.
4. Select **Finish**.

Next, select the source and the target of the transformation. The source of the transformation is the language-independent model (the file has an emx extension). Since the transformation creates a new model, you should create a new target

container (the file must also have an .emx extension). The wizard creates a .tc file. You will use the .tc file to launch the transformation.

To select the source and target, do the following:

1. Using the context menu, highlight the .tc file and select **Transform > COBOL Transform**. See Figure 11.

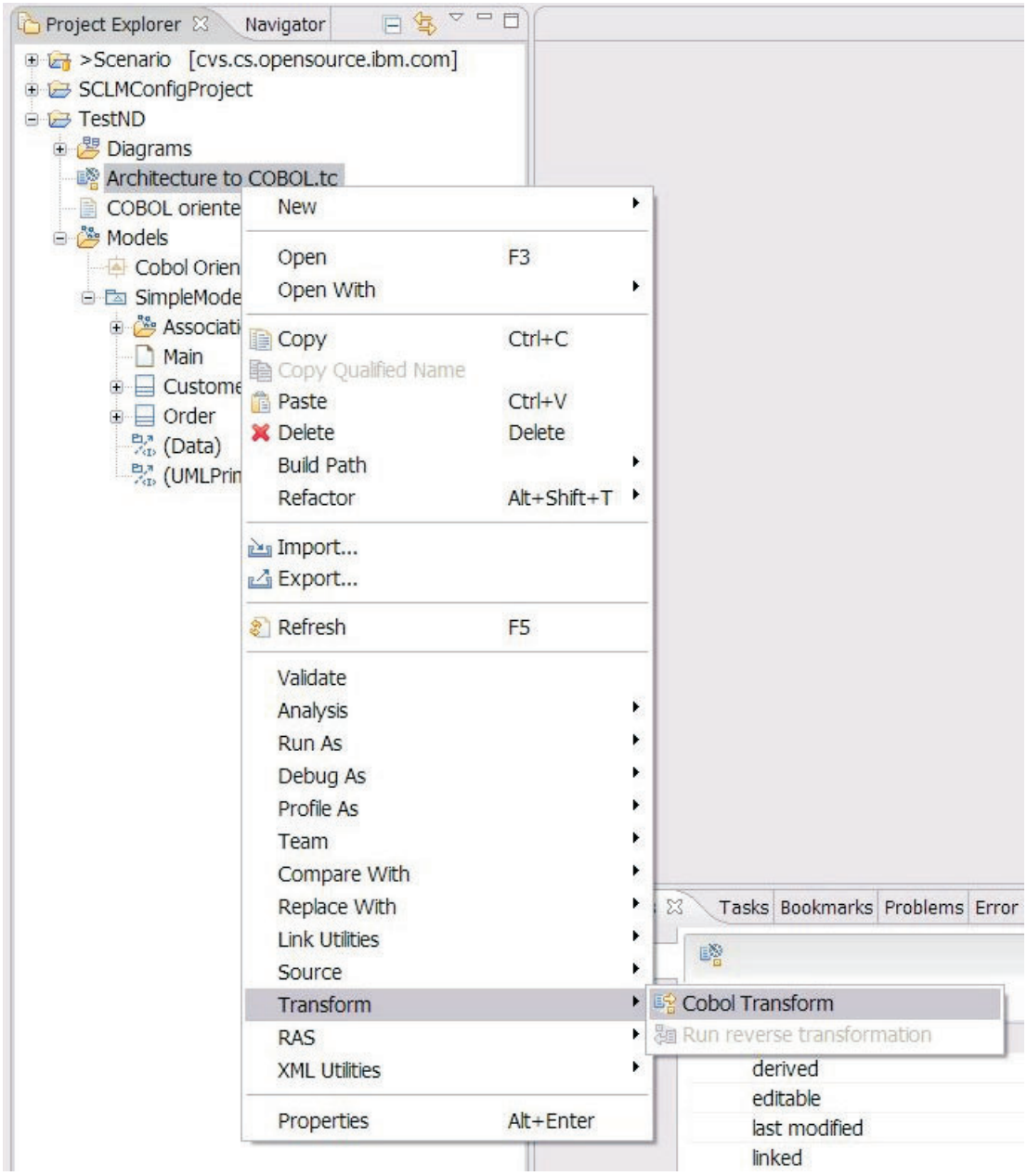

Figure 11. Launching the transformation

2. This creates the COBOL-dependent model. In the Project Explorer view under the Models section of your project, you will find the new model produced by the transformation.

The next step focuses on the transformed model.

Transformation to the COBOL-dependent layer

These sections describe the COBOL-dependent layer after transformation. The COBOL-dependent layer is then enhanced in the next step of this process.

Result of the transformation

The transformation has created a new model, transformed classes and their attributes, applied new stereotypes, and created a new diagram that contains the transformed elements of your initial diagram.

For reference, Figure 12 shows the model before transformation.

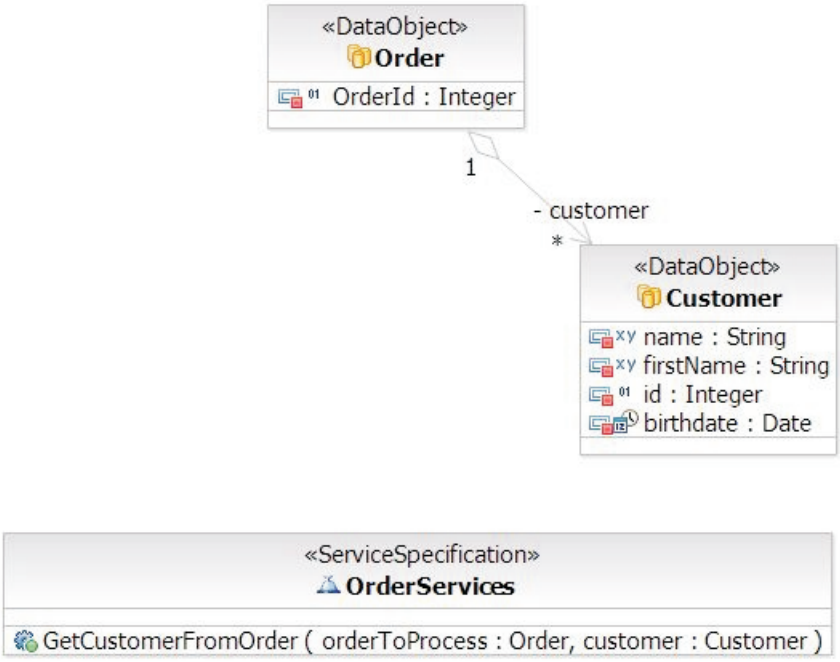

Figure 12. Simple model with primitive types

Figure 13 on page 13 shows the result of the transformation.

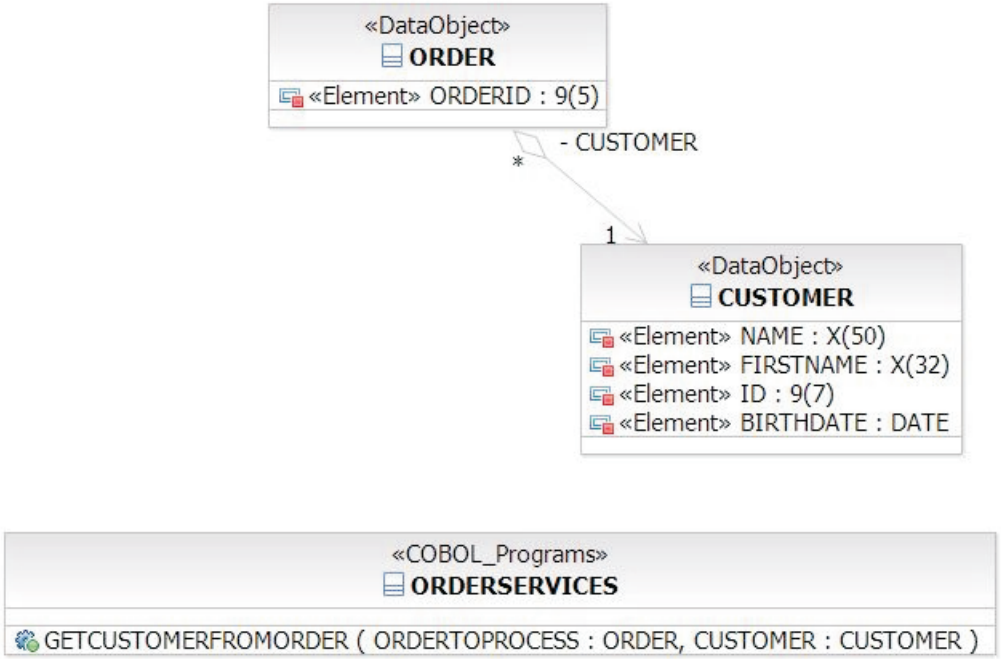

Figure 13. Result of the transformation

Renaming

Each component name has been renamed applying COBOL naming restrictions. The COBOL naming restrictions are the following:

- all names are uppercase.
- spaces are removed.
- forbidden characters (#, ? %, $) are translated to underscores.

Translation of the primitive types

New primitive types have been created to reflect the COBOL elementary definition of the attribute. The calculation is performed based on the primitive type (either UML default primitive types or the ones provided) and the extra characteristics of the (optional) stereotype.

Table 1 shows the translated primitive types where stereotypes are not applied.

Table 1. Mapping Primitive type to COBOL-dependent type

Primitive type	**COBOL-dependent type**
Binary	VARCHAR
Boolean	X
Date	DATE
Integer	S9(18)
Currency	S9(18)
Float	S9(16)V9(2)
Email	X(32)
String	X(32)

Translation of the attribute stereotype

The following primitive types are set if a stereotype from the data profile has been applied and some of its properties are set. In addition, the element stereotype from the COBOL profile is set on the attributes of the class.

The property usage is calculated based on the types as shown in Table 2.

Table 2. Property usage calculated on types

Stereotype	Property	Primitive Type Result	Usage (Element stereotype)
String	maxLength	X(maxLength)	display
Integer	digits, sign	9 (digits) or S9(digits) if digits < 18 X (digits) or X (digits + 1) if digits > 18	display
Float	digits, sign, decimals	S9(digits)V(decimals) or (digits)V(decimals)	comp-3
Email	displayName	X (length of displayName)	display
Currency	digits, sign, decimals	S9(digits)V(decimals) or (digits)V(decimals)	display
Boolean			display
Date			display

Translation of the class stereotype

The classes stereotyped as Data Object from the data profile become, in the new diagram, classes stereotyped with the Data Object of the COBOL profile. This Data Object stereotype of the COBOL profile contains a **toCopyBook** property that indicates whether a Copybook should be generated for data defined by the Data Object. For example, as shown in Figure 12 on page 12 and Figure 13 on page 13, classes stereotyped as "Service Specification" become in the new diagram classes stereotyped as "COBOL program."

Enhancement to the COBOL-dependent layer

These sections describe strategies for enhancing the COBOL-dependent layer in preparation for the next step in the process - generating source.

Working with the COBOL-dependent layer

COBOL-oriented UML artifacts are obtained from the transformation. There are several ways to enhance these artifacts before generation. The class diagram is a set of classes with relationships of various kind between these classes. COBOL data structure are expressed through these classes. The classes that drive the generation of data structure are the ones that are used as parameters of the program. See “Describing the program resources and linkage data” on page 24.

Refining the data definition for attributes

If the default translation is not satisfactory, you can edit the type and create the right one for an attribute. We recommend however that you change the available properties of the stereotype on the language neutral layer first (for example to set the length of a PIC X item) in order to capture more information on the top level layers.

Using the element stereotype for attributes

The element contains three properties that drive the generation:

- editedPicture - contains the COBOL edited picture
- clause usage - defines the usage clause value

- value - defines the initialValue.

Refining the relationships

UML relationships are translated during generation in order to create appropriate data definition. For some UML relationships, there are several strategies for translation that make sense. In this case, we have selected one default strategy for the UML relationship and introduced stereotyped relationships to express the other possible strategies. The following sections are the strategies used for the relationships.

Generalization (standard relationships): The COBOL data structure contains several groups, one for the attributes contained by the class that drives the generation and one for each of its ancestors. See Figure 14.

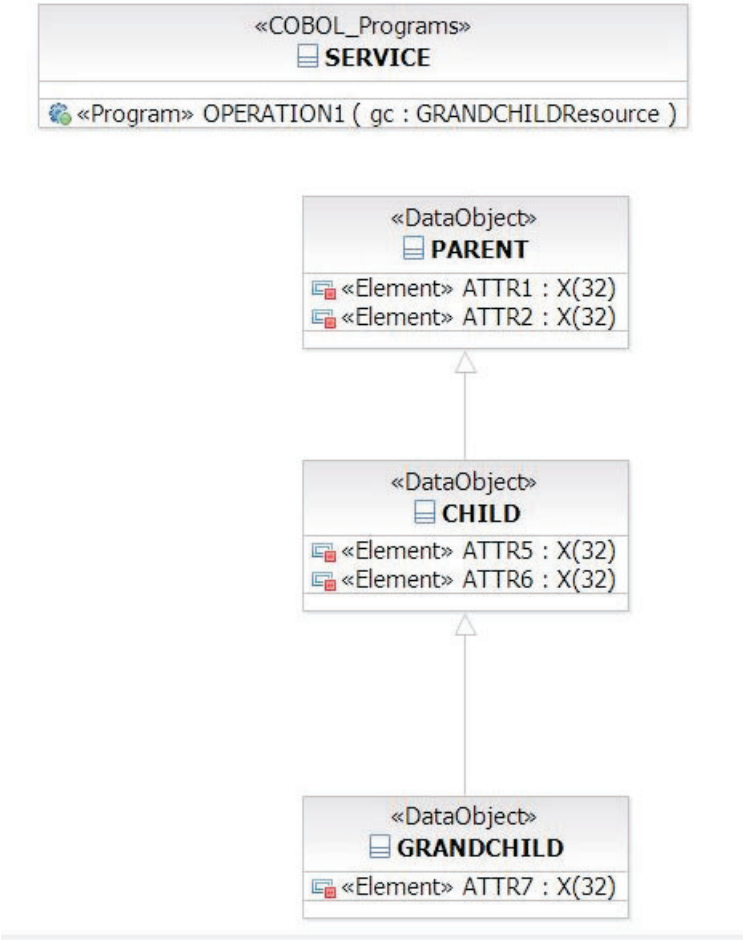

Figure 14. Standard relationships

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
```

Figure 15. Standard relationships shown as parent-child

Generalization with "Generalization" stereotype applied: The stereotype has a property called ″type″ that can be set to one of these 3 values:

- GroupAll - corresponds to the behavior described for the standard relationship
- GroupParent - groups are created for the parent classes only. The properties of all the classes that drive the generation are not embedded in a group.
- Flatten - no group is introduced. To avoid collision, data items are prefixed by the name of the class they belong to.

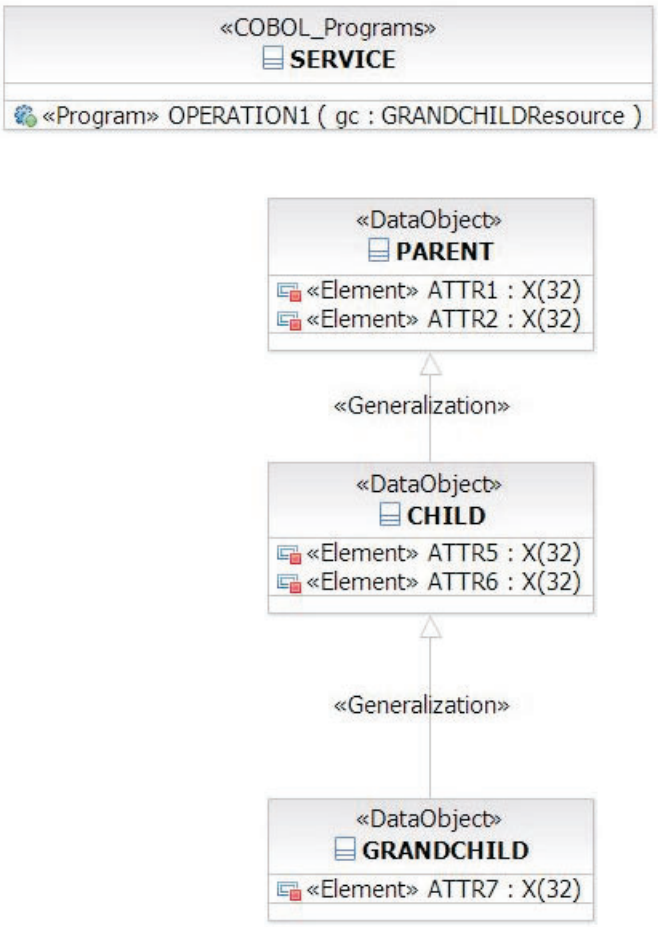

Figure 16. Applying the Generalization stereotype

The stereotype has type properties.

Figure 17. Stereotype type properties

By default, if "group all" is selected it behaves as the standard relationship.

GroupAll

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
```

Figure 18. Grouping by all

Group Parents

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
   05 ATTR7 PIC X(32).
```

Figure 19. Grouping by parents

Flatten

```
01 GRANDCHILD .
   05 PARENT-ATTR1 PIC X(32).
   05 PARENT-ATTR2 PIC X(32).
   05 CHILD-ATTR5 PIC X(32).
   05 CHILD-ATTR6 PIC X(32).
   05 ATTR7 PIC X(32).
```

Figure 20. Flattening relationships

Composition: The attributes of the class that are the target of a composition group are generated in a COBOL group related to the class that is the source of the composition link. The name of the group is the name of the target class.

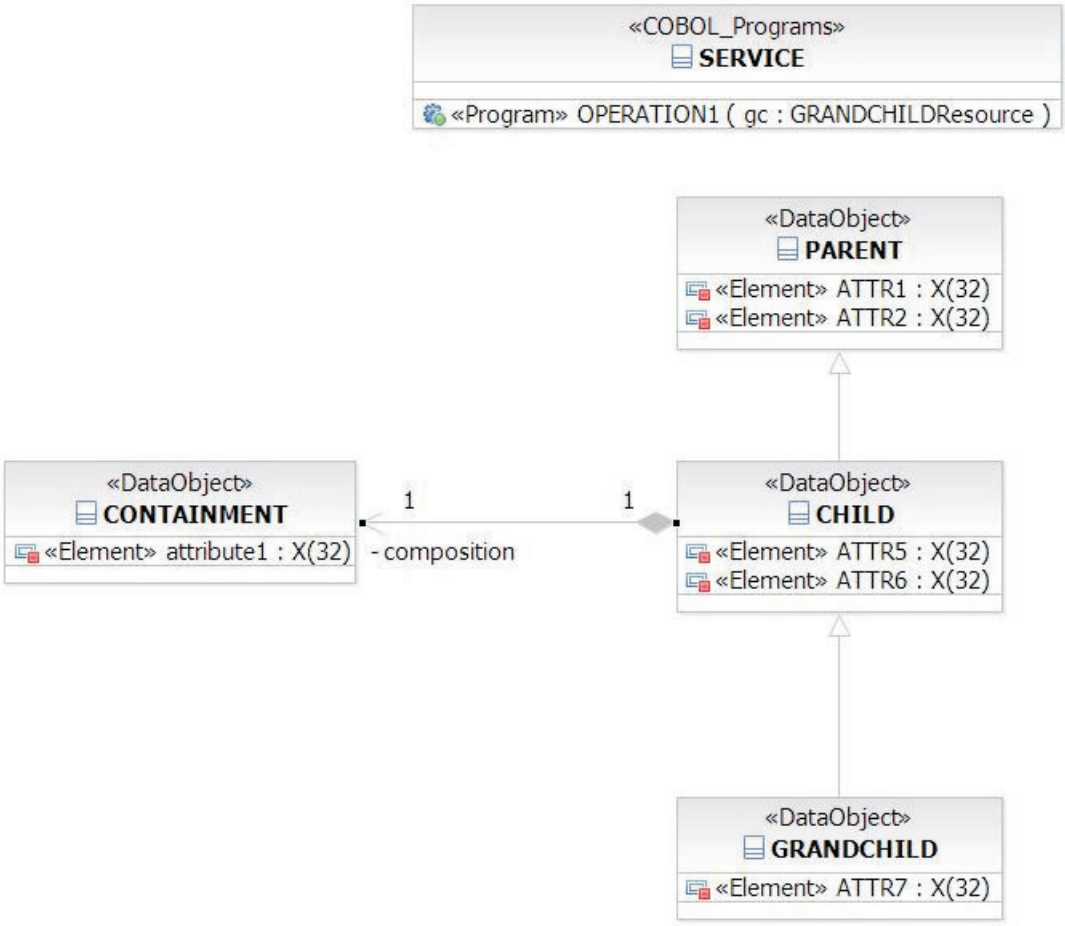

Figure 21. Composition

A composition relationship is established between two classes (Child and Containment in Figure 21). A group is created in the generated COBOL for this relationship (Figure 22).

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
      10 CONTAINMENT .
         15 ATTRIBUTE1 PIC X(32).
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
```

Figure 22. Composing relationships

Aggregation: A COBOL pointer is added to the COBOL group that is the source of an aggregation link. The name of the pointer is the name of the target class.

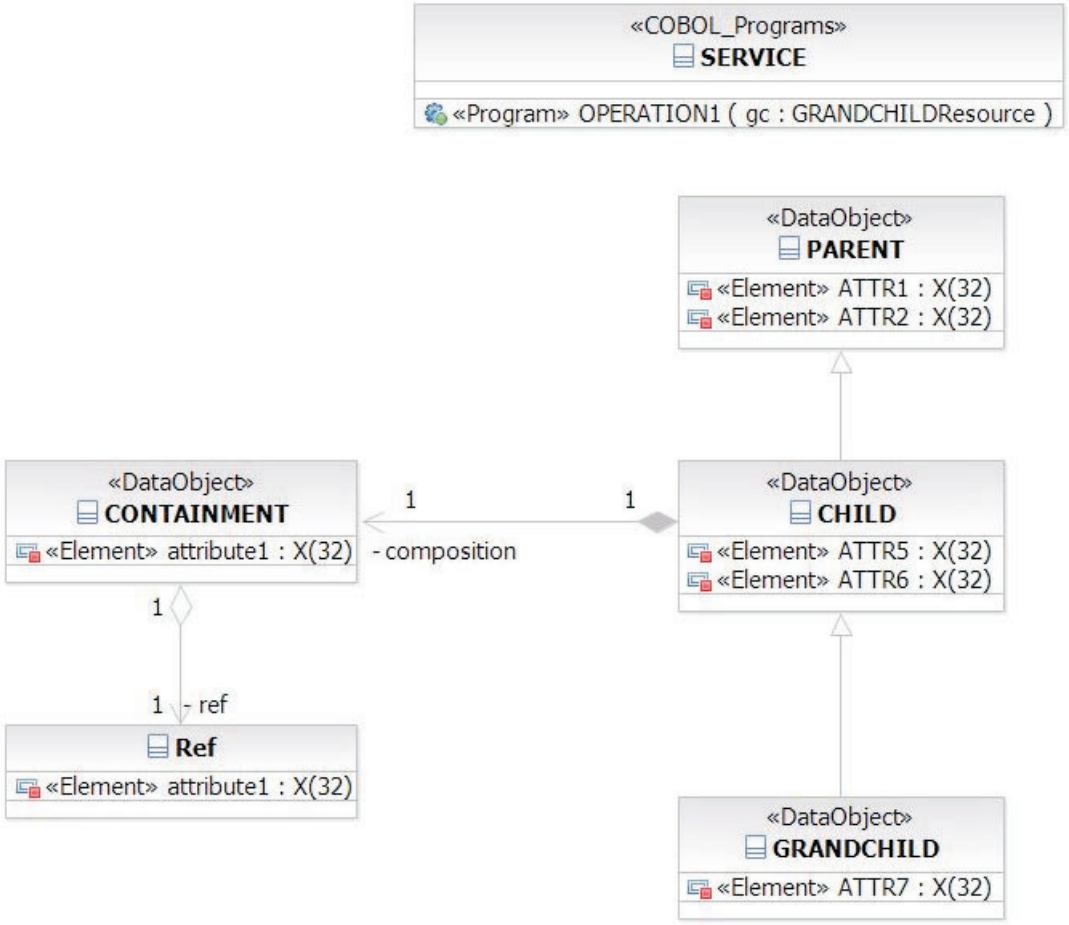

Figure 23. Aggregation

An aggregation is established between two classes (Containment and Ref in Figure 23). The following COBOL (Figure 24) will be generated from the diagram. Notice the pointer in the Containment group.

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
      10 CONTAINMENT .
         15 ATTRIBUTE1 PIC X(32).
         15 REF  USAGE POINTER.
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
```

Figure 24. Relationship aggregate

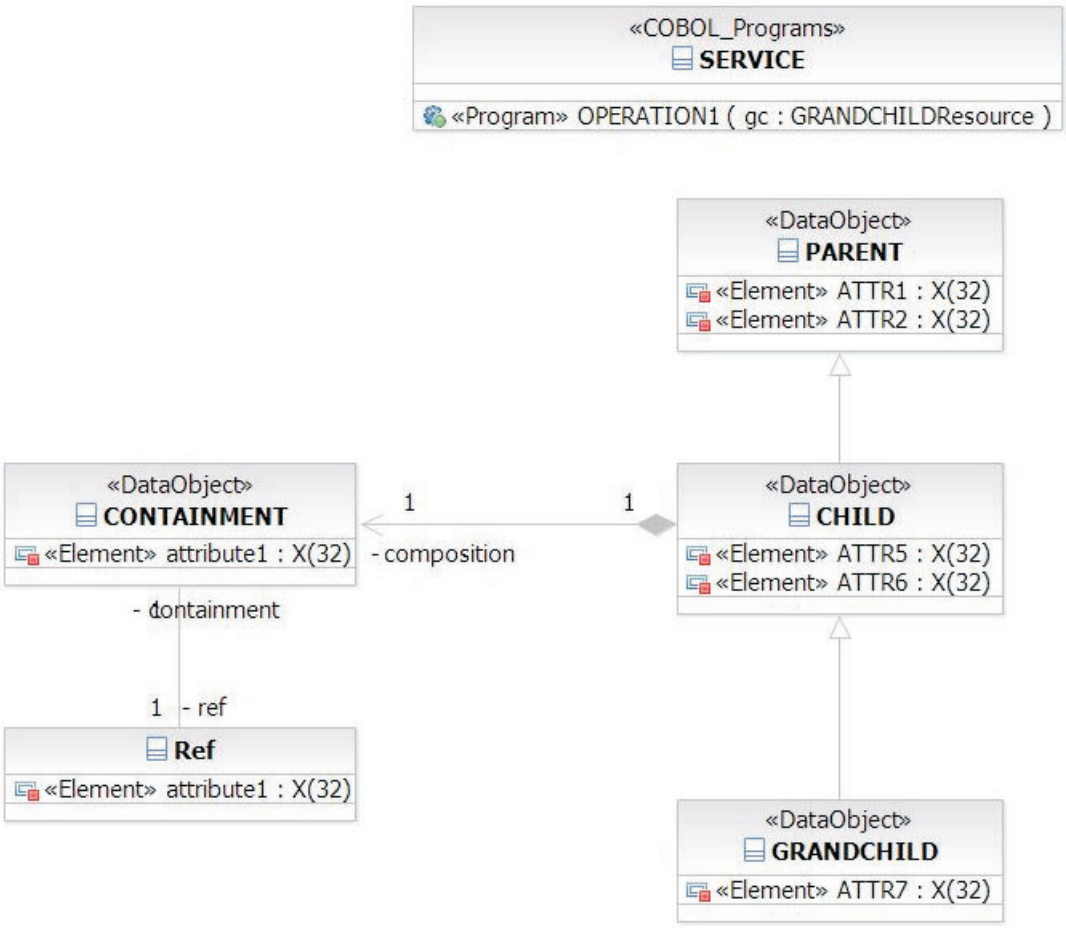

Figure 25. Association (between Containment and Ref classes)

In Figure 25, we have replaced the aggregation relationship between Containment and Ref by an association relationship. The result (Figure 26) is the same.

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
      10 CONTAINMENT .
         15 ATTRIBUTE1 PIC X(32).
         15 REF  USAGE POINTER.
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
```

Figure 26. Aggregation as parent-child

Association with the "Redefines" stereotype applied: The target of such an association generates a COBOL group that redefines the contents of the source of

the relationship.

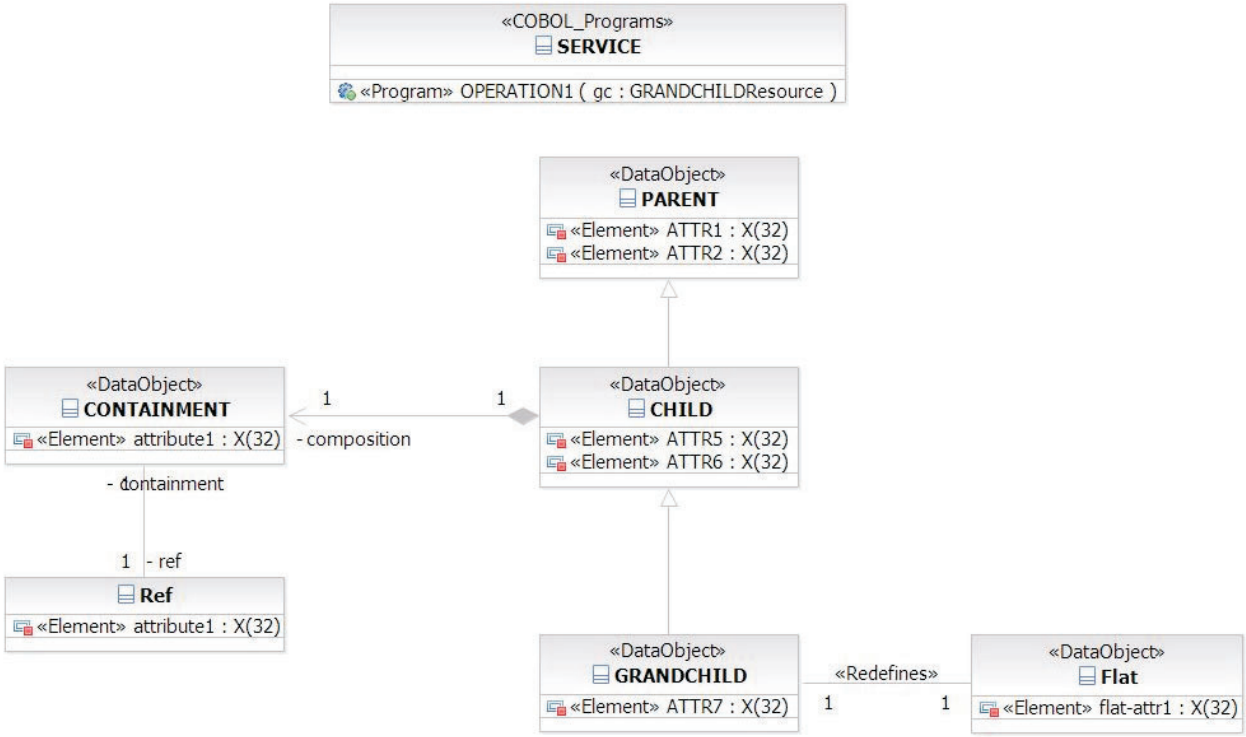

Figure 27. Stereotyped Association - Redefines

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
      10 CONTAINMENT .
         15 ATTRIBUTE1 PIC X(32).
         15 REF  USAGE POINTER.
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
   05 FLAT redefines GRANDCHILD.
      10 FLAT-ATTR1 PIC X(32).
```

Figure 28. Generated COBOL that translates the redefines relationship

The Redefines association can be used at different places in the hierarchy of a class. Figure 29 on page 23 is an example of the redefines association that is used in two places of the hierarchy. Figure 30 on page 23 displays the result in COBOL.

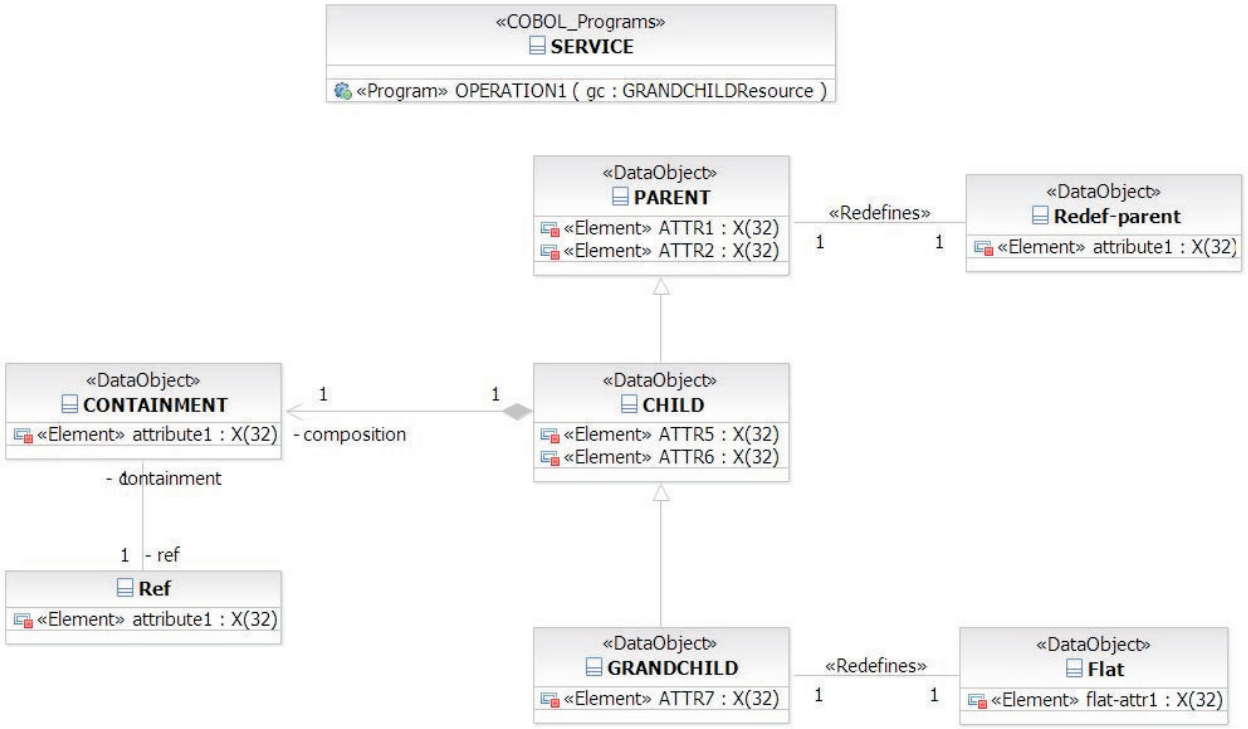

Figure 29. Redefining association

```
01 GRANDCHILD .
   05 PARENT .
      10 ATTR1 PIC X(32).
      10 ATTR2 PIC X(32).
   05 REDEF-PARENT redefines PARENT.
      10 ATTRIBUTE1 PIC X(32).
   05 CHILD .
      10 ATTR5 PIC X(32).
      10 ATTR6 PIC X(32).
      10 CONTAINMENT .
         15 ATTRIBUTE1 PIC X(32).
         15 REF  USAGE POINTER.
   05 GRANDCHILD .
      10 ATTR7 PIC X(32).
   05 FLAT redefines GRANDCHILD.
      10 FLAT-ATTR1 PIC X(32).
```

Figure 30. Redefining parent-child

Combination of these relationships (stereotyped or not) can be used in the class diagram in order to describe the appropriate data structures.

Note: If you want to redefine the whole Data Object and its ancestor you should use the ″flatten″ strategy for inheritance.

Describing the program resources and linkage data

The operation of a program contains input and output parameters based on classes defined in the model. These parameters are appropriate to identify the external data manipulated by the program. The COBOL profile creates a **Resource Stereotype** to identify resources manipulated by a program, such as a file, a message queue, or a relational table. In order to define a resource, you need to create a dedicated resource class that references a root Data Object to define the data of the resource.

The relationship between the Resource class and the DataObject class shows that the Resource's contents is defined by the Data Object. This relationship is captured by a Stereotype. Once the Resource class is defined and connected to the appropriate DataObject, the input and output parameters can target this resource class instead of the DataObject class. You can define the model to manipulate resources through a parameter which type is a Resource stereotyped class. You can define the model to manipulate linkage data through a parameter which type is a DataObject stereotyped class. To set a program to manipulate the resource, do the following:

1. Create a resource class that reference the DataObject that will be manipulated by the resource
2. Reference the DataObject from the resource.
3. Apply the ResourceIsStructured stereotype to the relationship.
4. Change the operation parameter so that its type targets the Resource class and not the DataObject class. If an operation has a parameter whose type is directly on DataObject, the generation assumes that it is a data definition to be used in the linkage section.

In order to automate this process, a RSA pattern is provided that does these steps automatically.

To enhance the diagram, do the following.

1. From the Workbench, select **Window > Show View > Other...** and then select the **Pattern Explorer** view. See Figure 31 on page 25.

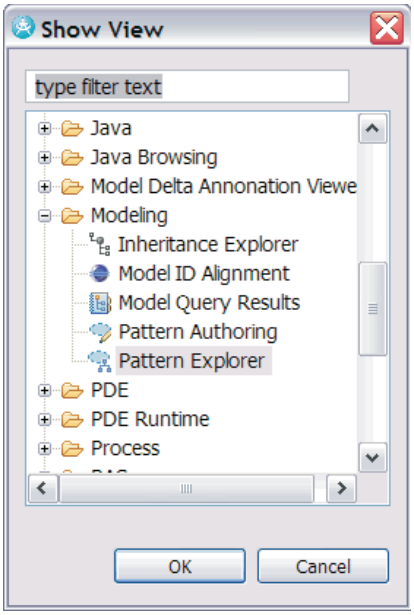

Figure 31. The Show View window

2. The **Pattern Explorer** view opens.
3. Select the **COBOL program** pattern under **UML to COBOL** as shown in Figure 32.

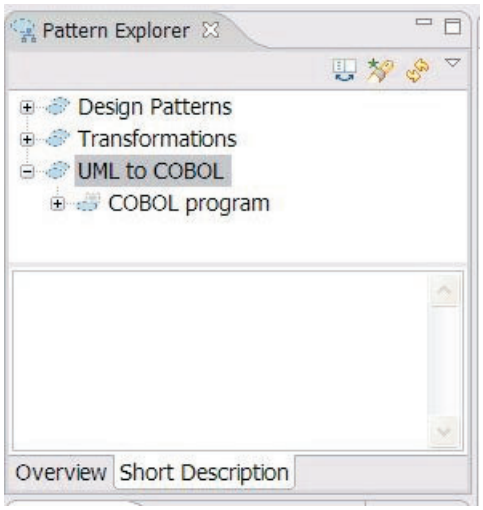

Figure 32. Pattern Explorer view

4. From the Pattern Explorer view, drag the **COBOL program** to the diagram and drop.

Figure 33 on page 26 shows the diagram result of the drop and drag.

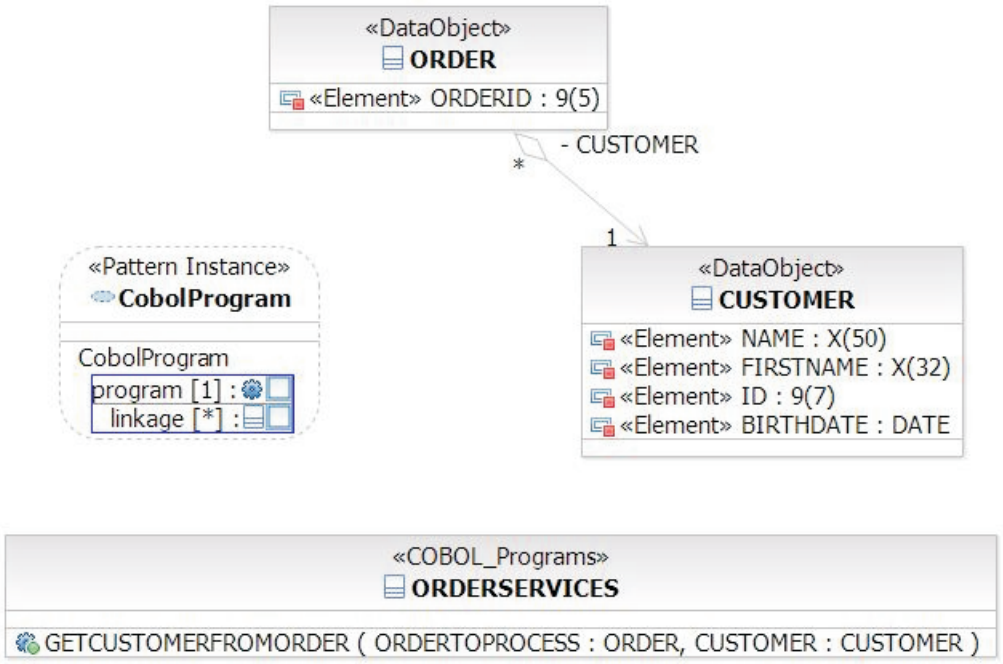

Figure 33. Drop and drag to the diagram

The first parameter of the pattern is the operation that you want to process. Drag and drop the operation of the COBOL_Programs class that you want to transform into a program. The pattern now automatically performs changes against the parameters of the operation - all its parameters are modified and resources are created. You can then specify which Data Object you want to use in this program as linkage section in the second parameter. The changes are directly visible on the operation parameters. To see the newly-created Resources in your diagram, drag and drop the resource classes from the Project Explorer to the diagram.

Figure 34 on page 27 shows the result of applying the COBOL program pattern and adding ResourceDescription.

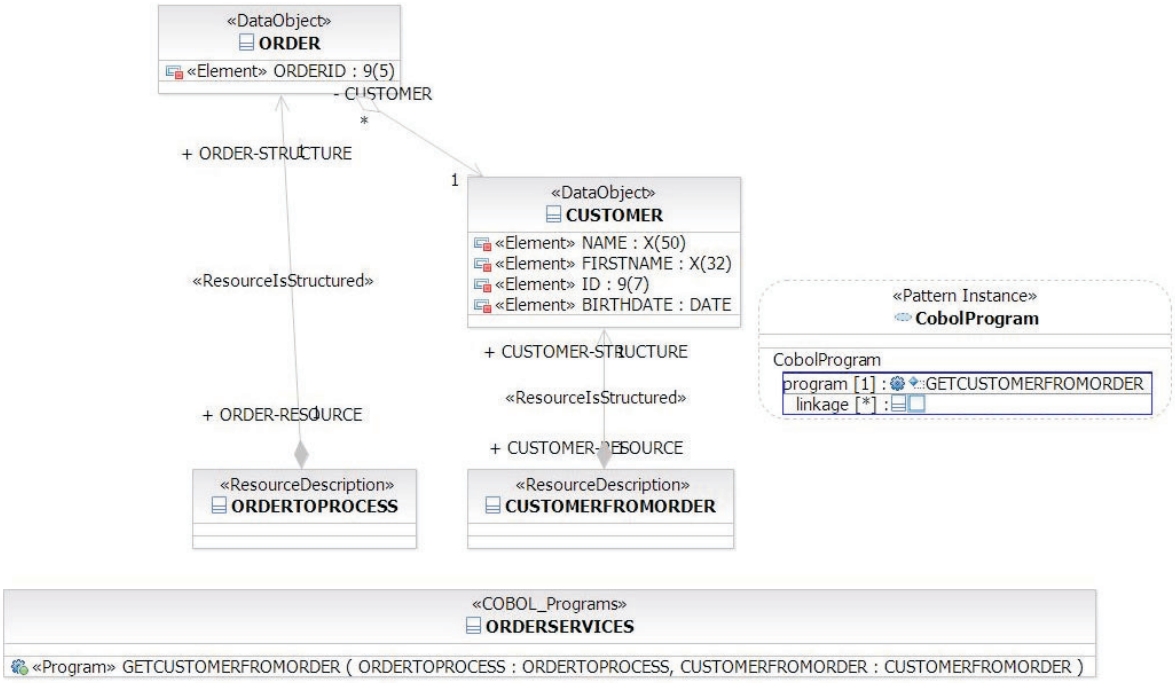

Figure 34. Results

It is a good practice to create several diagrams that can display subsets of your models; for example, a separate diagram dedicated to the program, one to its resources, and one to the root definition.

Describing a basic control flow of a program

You can associate an activity diagram to an operation. Since the operation represents the program (or an entry point of a program), this is how you describe some elements of the logic of the program. Do the following:

1. In the Project Explorer view, use the context menu to highlight the operation and select **Add UML > Activity Method**.

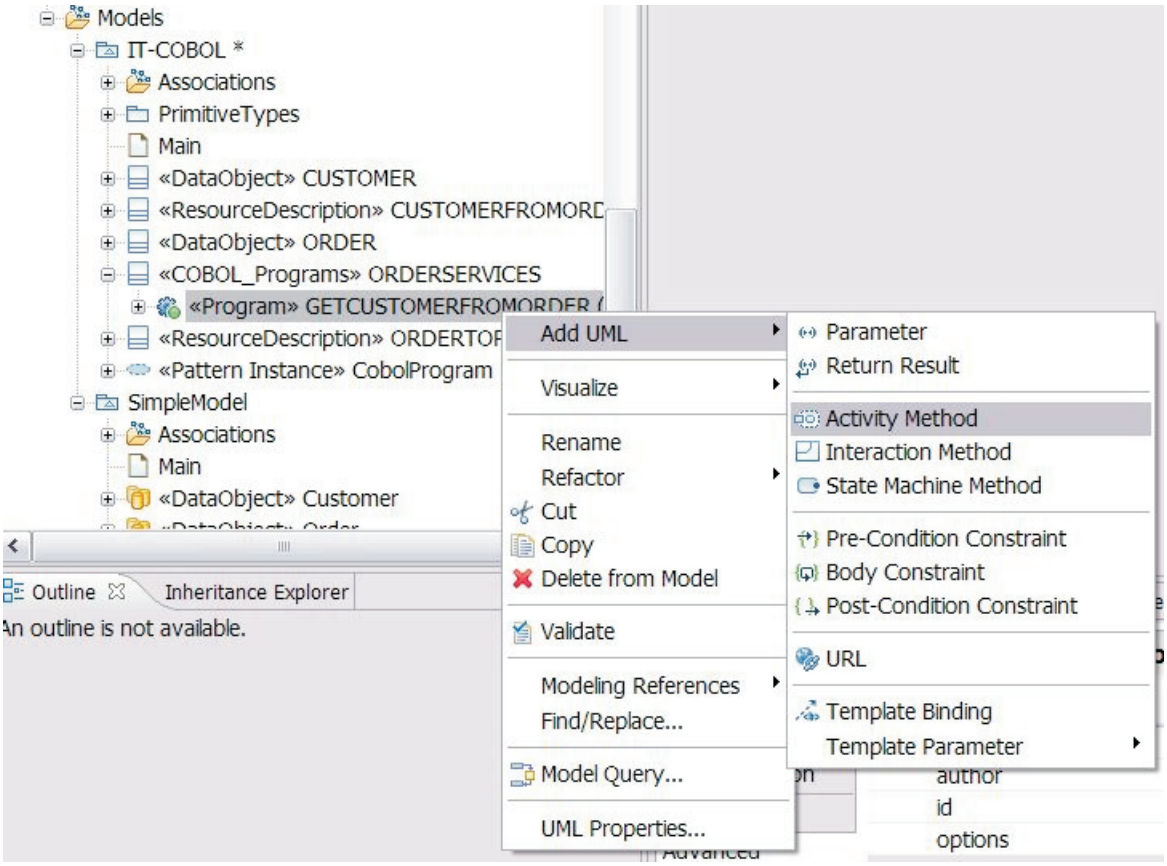

Figure 35. Associating an activity

2. To add the activity, first open the Apply Stereotypes window.
3. A few stereotypes for basic high level functions that a program can perform are already provided. First add the COBOL procedure profile to your model. Figure 36 on page 29 shows the provided stereotypes.

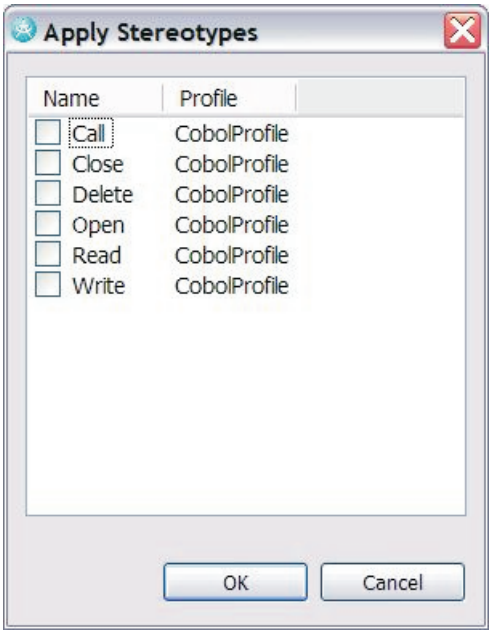

Figure 36. Apply Stereotypes for activity diagram

4. Select **Ok**.

Figure 37 shows an activity diagram of a simple open-and-close operation.

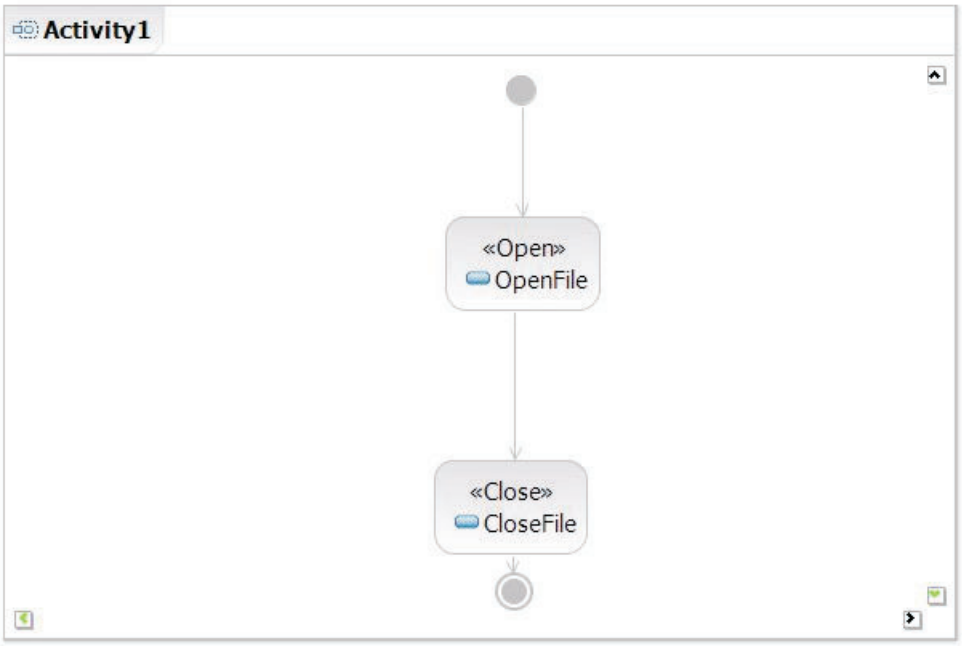

Figure 37. An example of a simple activity.

Note: Refer to RSA documentation for general information about RSA modeling.

You can indicate which resource the activity manipulates through dedicated properties of the stereotype. Figure 38 on page 30 shows the property window of the selected node.

Figure 38. Properties window of the selected node

Limitations

There are some limitations in the tech preview during the generation of the activity diagram. Some advanced scenarios are not supported yet:

- You cannot merge several nodes directly to an activity final node. You need to merge them first in a node that you then connect to the final node.
- Branches and merge are supported to a certain extend only. Complex cases that involve unbalanced branches and merge may fail.
- Loops are not correctly captured.

Generating COBOL Source

The final goal of the modeling transformations is to generate COBOL source code that can be enhanced further within the development environment of Rational Developer for System z.

At this part of the process, the system architect who has been modeling the programs and data structures has been using the capabilities of Rational System Architect, enhanced with profiles containing additional stereotypes and patterns. The second modeling transformation generates the output that is used with RDz to continue COBOL development for System z.

One COBOL source program is generated for each modeled operation in the COBOL-specific model that has been used with the COBOL program pattern. In addition, a Copybook is generated for each defined data structure that has the DataObject stereotype applied, and the **toCopybook** attribute of the stereotype set to true.

To set the **toCopybook** attribute, do the following:

1. Select the node in the diagram.
2. Open the **Properties** window view.
3. Under **Stereotype Properties**, select the **DataObject** property drop-down menu.
4. Set **toCopybook** to True.

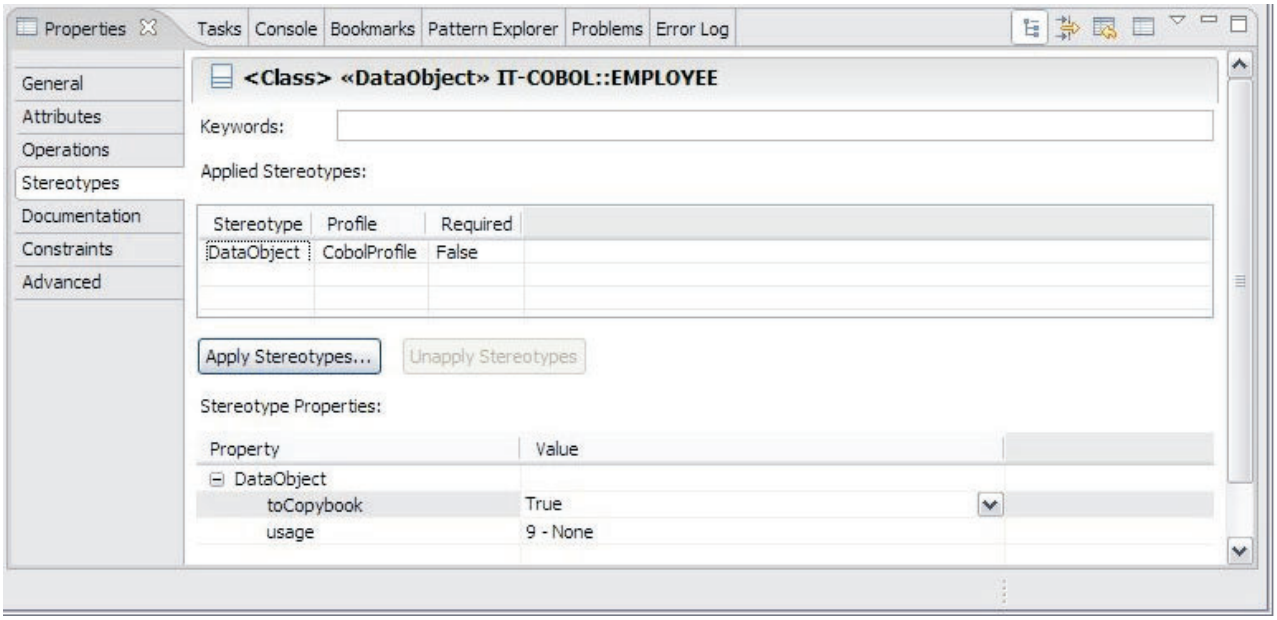

Figure 39. Setting the toCopybook attribute

If the **toCopybook** attribute is set to false, the data structure is declared in each program as needed.

Generating from RSA

The second transformation, using the COBOL-specific model as the source, produces an intermediate file, with an extension of .cobolumlmodel. The transformation definition created earlier (in the .tc file) declares what file is the transformation target.

This intermediate file contains an EMF representation of the data structures and programs being modeled, and it can be shared (for example, by version control system, email) with an RDz user who can use the file to generate COBOL source (see "Generating from RDz").

However, if the system architect using RSA to model the COBOL programs also has RDz installed in the same package group as RSA, the COBOL source is generated by the second transformation along with the target .cobolumlmodel file. The COBOL source is placed into the same modeling project as the language-independent and COBOL-specific models, and can then be moved into any COBOL development project using the standard workbench tools.

Generating from RDz

If the system architect who modeled the COBOL programs does not also have RDz installed, the final output of the modeling transformations is the intermediate file with extension .cobolumlmodel . This file is then shared with a developer who has RDz installed to complete the final transformation into COBOL source code. The final transformation can be invoked in a couple of different ways. The following sections describe some of these ways.

Using the new COBOL Project wizard

The user can create a new COBOL project in their workspace that contains the generated source code from the UML-to-COBOL transformation. Do the following:

1. Select **New > COBOL Project**

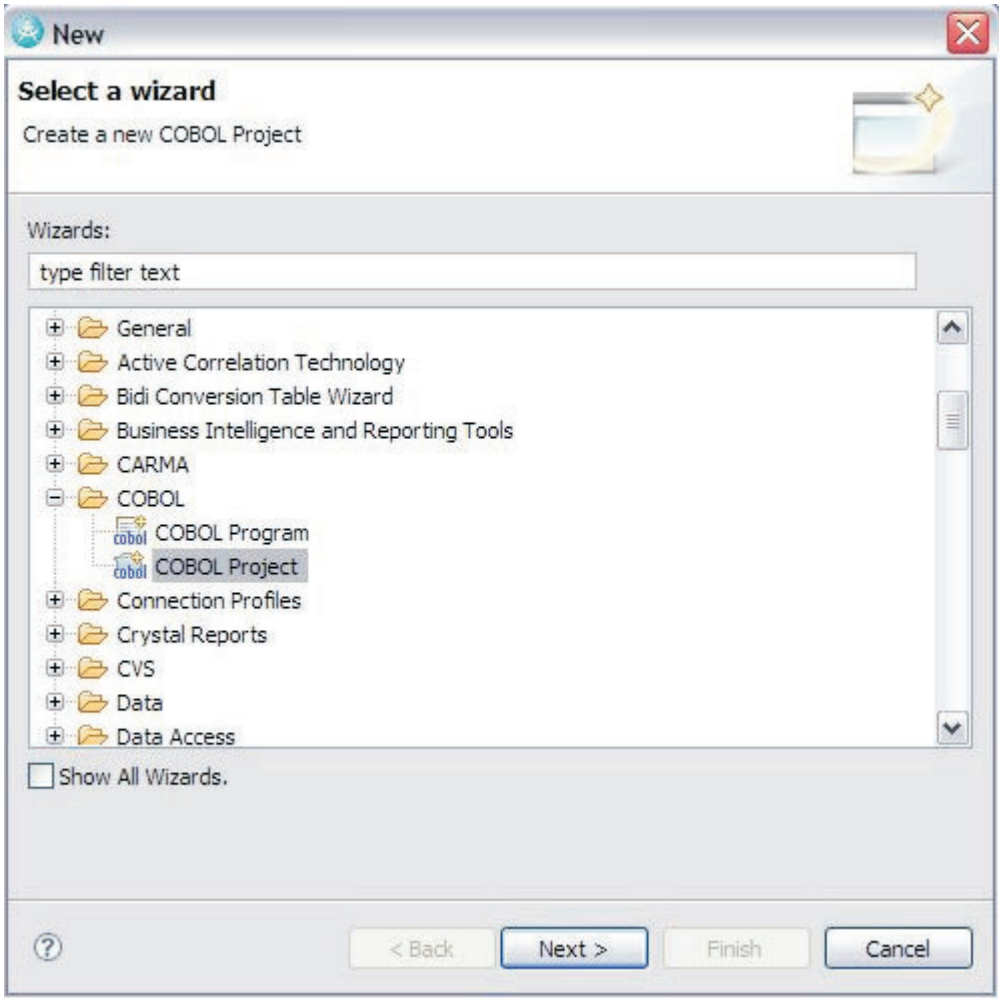

Figure 40. New project window opens

2. Complete the new project name and location, and select whether you want to use the Remote Synchronization feature, then select **Next**.
3. Select the **UML-to-COBOL Generation** application template, then select **Next**.

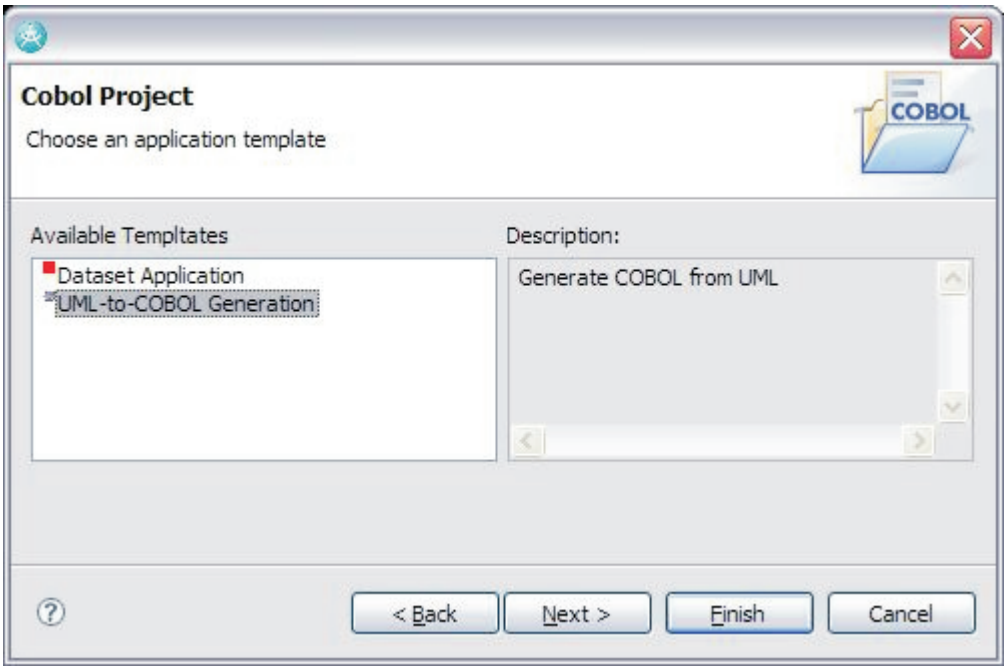

Figure 41. Select UML-to-COBOL Generation

4. Enter or select the file name of the intermediate file to use to generate the code. Any file location accessible by the workstation is accepted, the file does not have to be first imported into the workspace.

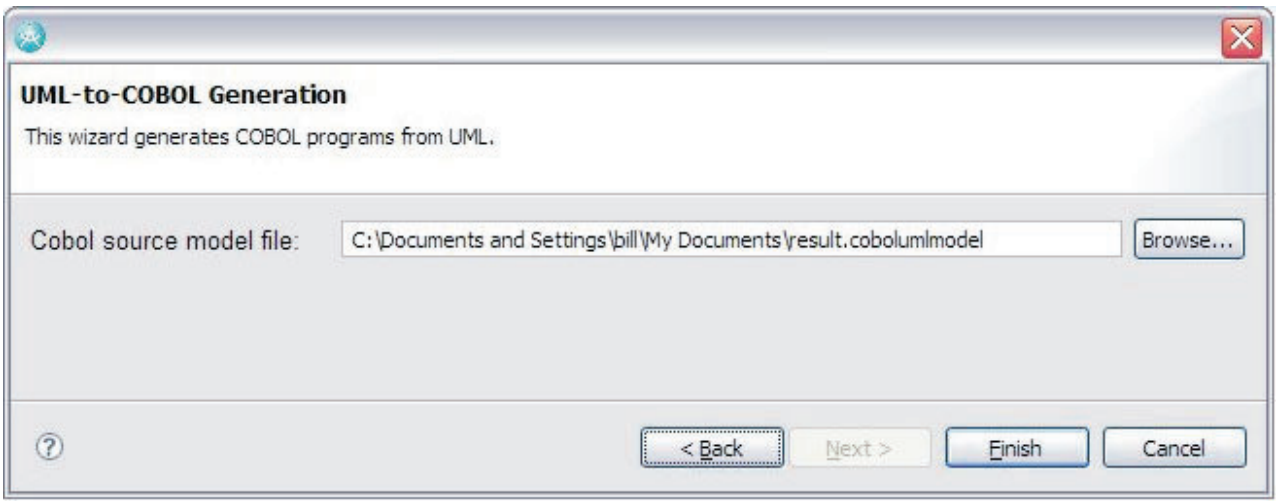

Figure 42. UML-to-COBOL Generation window

The project is then created. The COBOL source code generated from the intermediate file which is also copied into the new project. Selecting the Remote Synchronization feature in the wizard in order to integrate with System z is recommended. The generated source code can also be copied or moved into a z/OS® Project for integrating with System z.

Using the context menu

An alternate way to generate COBOL based on a COBOL source model file that is already in the workspace is to use the file's context menu (right-click the mouse over the file in the **Project Explorer** view).

1. Select the **Generate COBOL** menu item from the file's context menu.
2. Choose whether to create a new project or generate the COBOL to the same project where the source mode file is located.

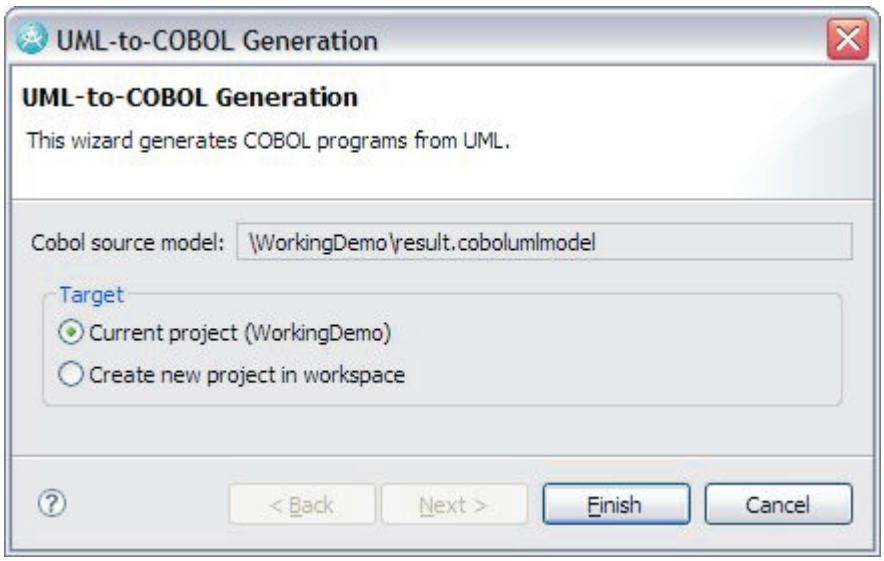

Figure 43. UML-to-COBOL Wizard window

3. If selecting to create a new project, then enter the project name and location, and select whether you want to use the Remote Synchronization feature (like above when using the **New COBOL Project** wizard with the UML to COBOL template), then select **Finish**.

Chapter 3. Advanced COBOL Editor Features

The Advanced COBOL Development technology preview provides expanded features and capability for the COBOL editor such as real-time syntactic and semantic checks on the source, refactoring actions, and actions and views that help navigate source code.

The Advanced COBOL Editor allows you to partially validate your code without compilation. You can also move to the declaration of any data item in the program file by using the **Open Declaration** context menu action. You can reliably change the name of a data item in a single file with the preview's new refactoring capabilities. The ability to automatically remove noise words from your program is also available as is a new **Perform Hierarchy** action that displays your program's perform paragraph calls in a tree view format.

Real Time Validation

A lightweight parser is dispatched when the source is changed in order to perform syntactic checks on the COBOL source. As you edit, warning symbols appear in the vertical ruler located along the left margin. Move your mouse over the warning symbols to see text that explains the problem. An abstract syntax tree is subsequently analyzed to resolve references to data items, paragraphs, and sections. A warning symbol appears in the vertical ruler if a reference can not be resolved. For example the statement "PERFORM MYPARAGRAPH" is flagged with a warning if the paragraph "MYPARAGRAPH" does not exist.

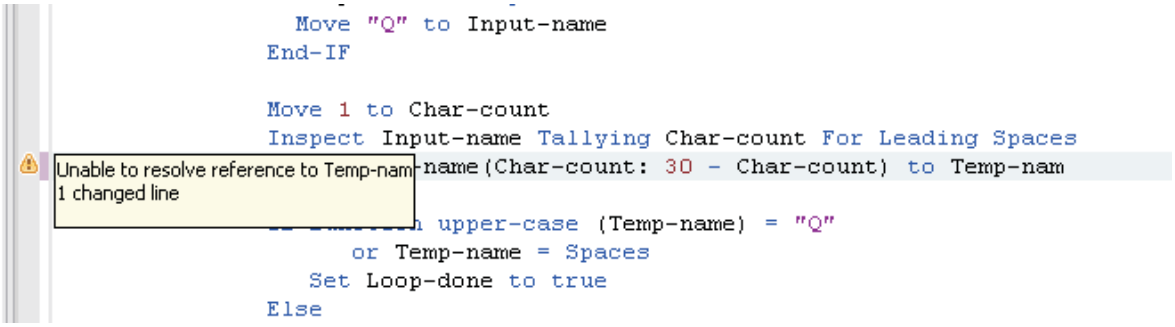

Figure 44. Statement is flagged with a warning

Open Declaration

The **Open Declaration** action allows you to navigate from a reference to a declaration. You can select this action from the context menu. It can also be selected and mapped to a keybinding by selecting the **Windows -> Preferences -> LPEX Editor -> User Key Actions** menu.

Using the mouse, the user first selects or highlights a reference to a data item, paragraph, or section, then selects **Open Declaration** in the context menu (mouse right-click). The action moves the text selection in the editor from the selected reference to its corresponding declaration. The user can navigate between the previous position using the navigation arrows on the toolbar or typing the "Alt-Back Arrow" keybinding (or "Alt-Forward Arrow").

Step1

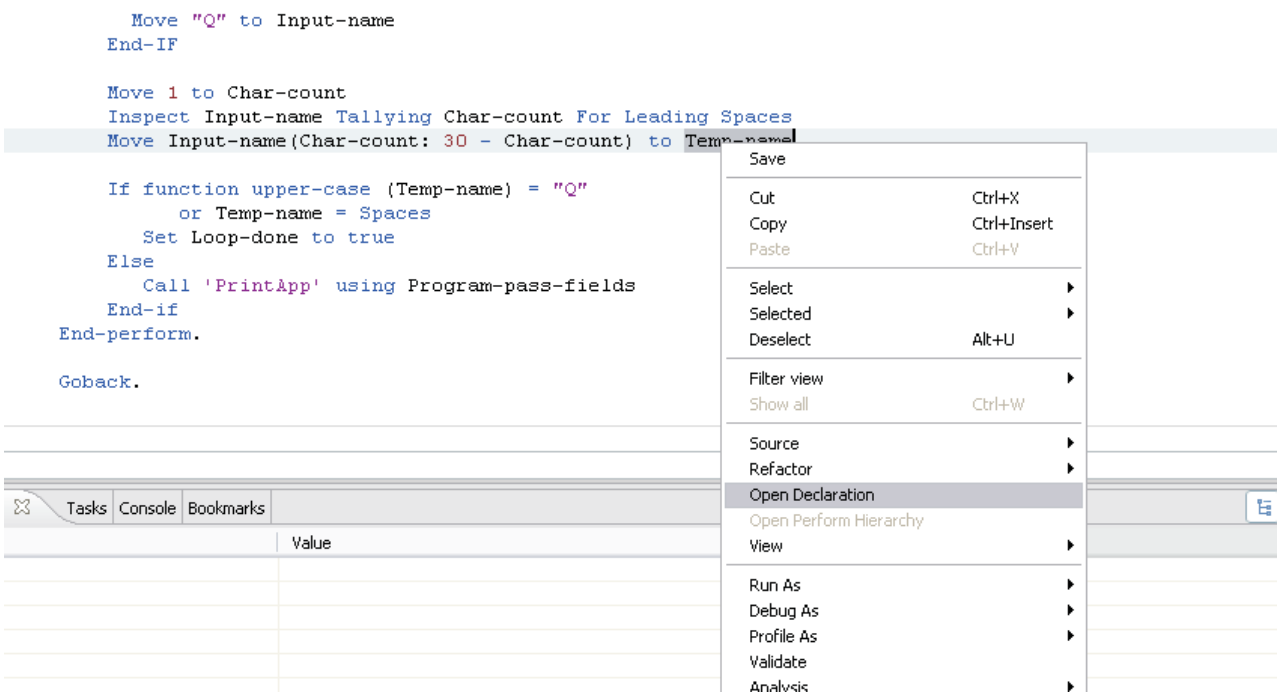

Figure 45. Open Declaration step 1

Step2

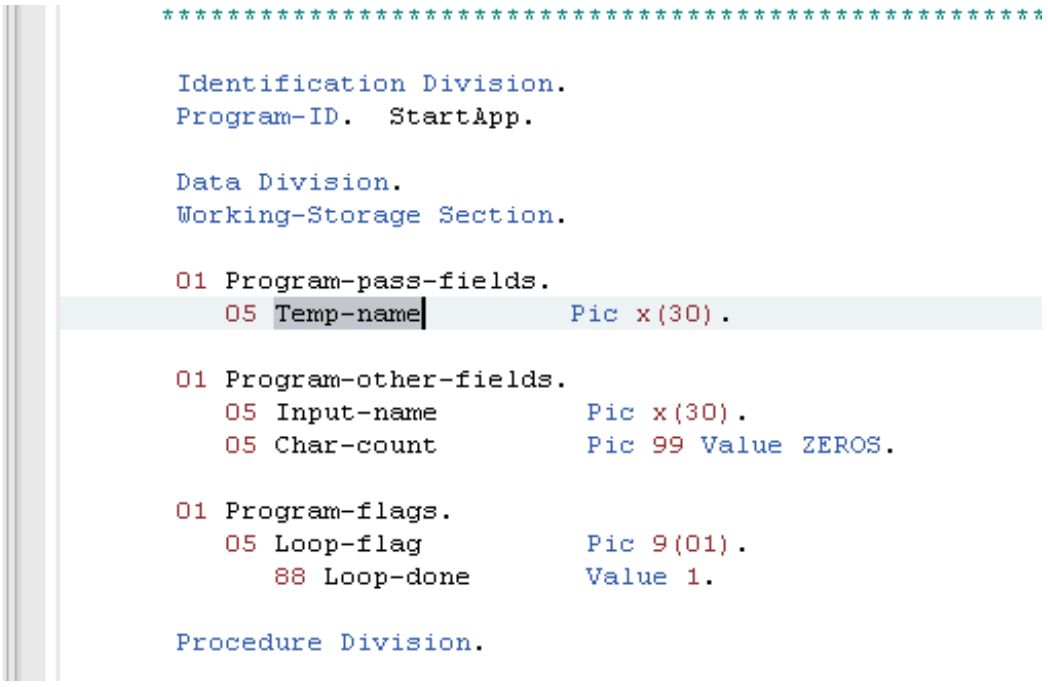

Figure 46. Opening Declaration step 2

Refactor

This technology preview provides the following two refactor operations:

- **Rename** - renames any data item or paragraph.
- **Remove Noise Words** - removes selected unnecessary words from the COBOL source.

The **Rename** shows an information screen with the projected warnings if the refactor is committed as well as how many data items are going to be changed. Optionally, another page is shown with a preview of what the source will look like once the refactor is accepted compared against the original source.

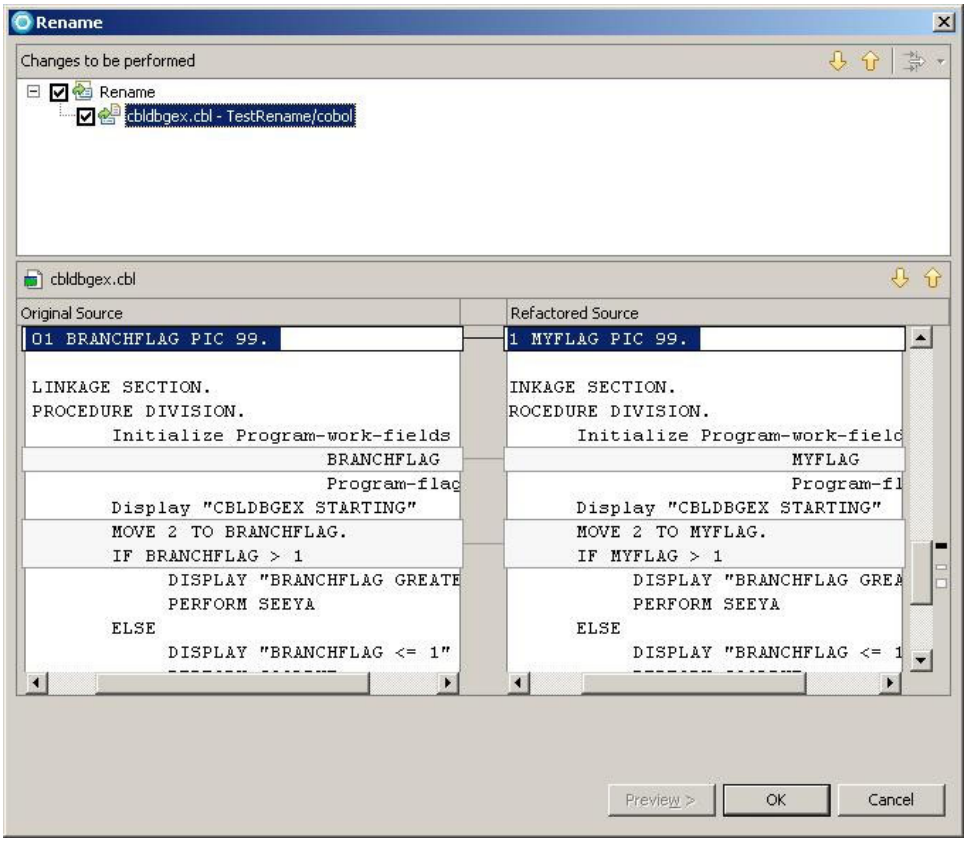

Figure 47. Preview page

Perform Hierarchy

When a reference to a paragraph or section is selected, the user can select the **Open Perform Hierarchy** action from the context menu. The **Perform Hierarchy** view displays the paragraphs and line numbers for each statement in the hierarchy and allows you to easily navigate among those statements. The **Perform Hierarchy** view has two modes: the "Performer Hierarchy" and the "Performee Hierarchy." The **Performer Hierarchy** displays the references to the selected paragraph. For example, given the paragraph "Copy-input-to-output," the **Performer Hierarchy** displays all paragraphs and sections that can transfer control to that paragraph with a statements such as "Perform Copy-input-to-output." The **Performee Hierarchy** displays the references to other paragraphs contained within the selected paragraph. For example, given the "Copy-input-to-output" paragraph, the **Performee Hierarchy** displays a statement that can transfer control outside of the

selection paragraph with statements such as ″Perform Read-next-input-data.″ Since perform hierarchies can be nested arbitrarily, the **Perform Hierarchy** is a tree view that allows you to traverse this nesting.

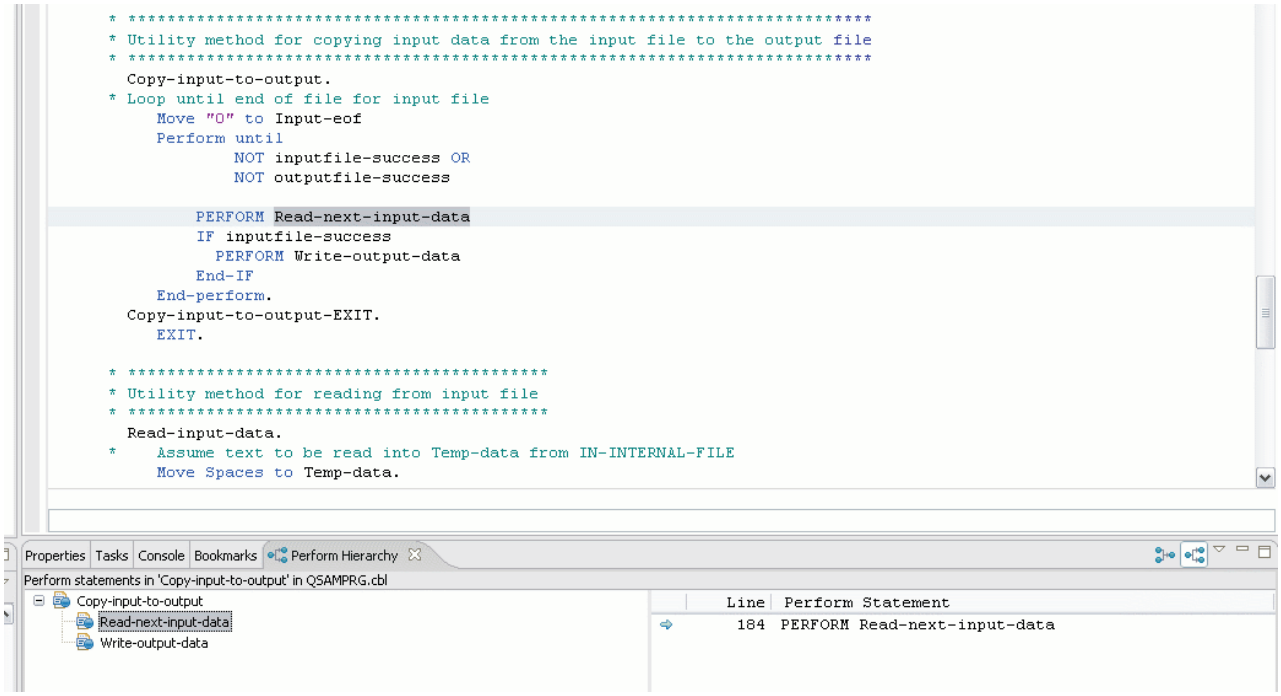

Figure 48. Example of Performee Hierarchy

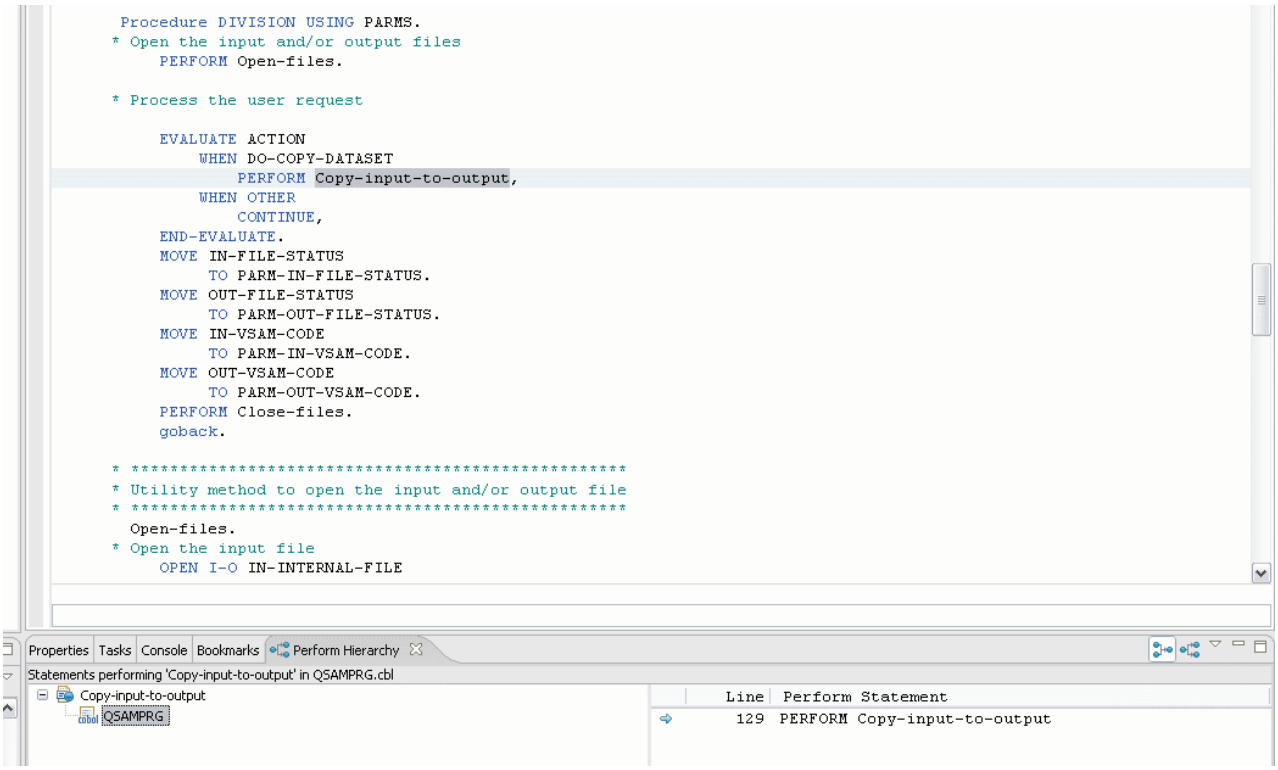

Figure 49. Example of Performer Hierarchy

Additional Information

A viewlet tutorial detailing current editing features is available on the Technology Preview web site.

Limitations

The advanced COBOL Editor technology preview has the following limitations:

- The Editor does not parse CICS® and SQL in order to check for correct syntax. The parser does not flag EXEC ... END-EXEC statements as errors but does not attempt to parse the contents of these statements. Errors are not reported for invalid CICS or SQL syntax and they are not processed during semantic analyses.
- The Editor does not semantically process COPY statements. This means that although the Editor correctly parses a COPY statement, it does not include the contents of the copybook when performing semantic analyses such as reference resolution. Errors are reported if you reference a data item declared in a COPYBOOK.
- **Refactor** operations are not atomic and cannot be undone.
- The **Perform Hierarchy** will not be in sync if the source file is modified or closed.

If the above limitations result in incorrect behavior, the user can optionally enable or disable these features under the preference page at **Window > Preferences > COBOL > Tech-Preview**. From the menu you can:

- Select **Enable editor annotations and tooling** to receive real-time syntax checking of COBOL files via warning annotations in the editor.

- Select **Enable tooling only** to disable editor annotations but leave the tech-preview actions in the context-menu enabled; for example **Open Declaration** or **Refactor**.
- Select **Disable editor annotations and tooling** to disable both the editor annotations and context-menu actions.

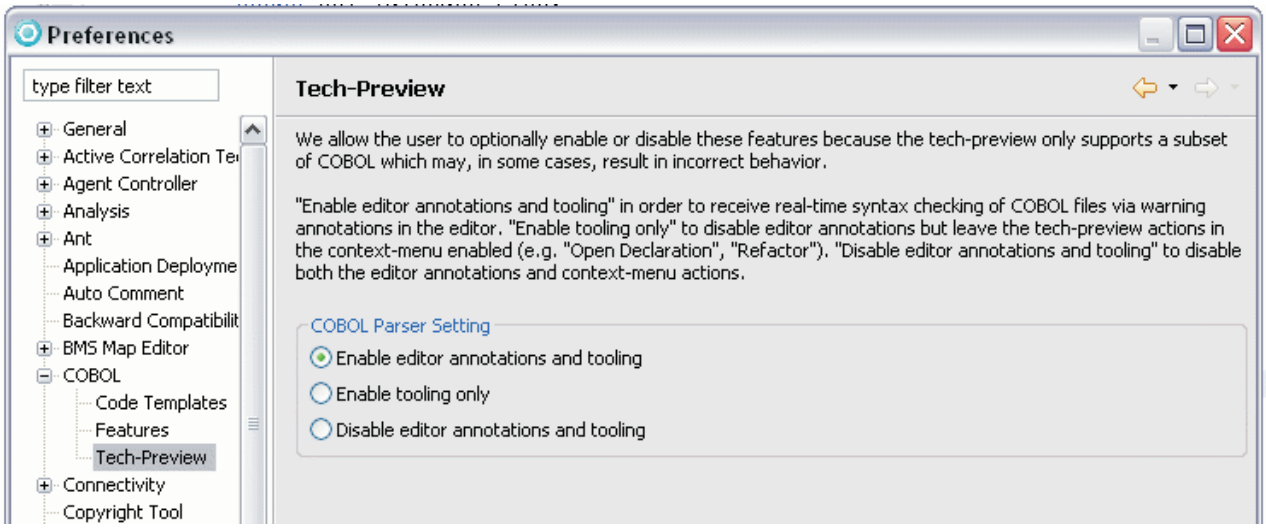

Figure 50. The Tech-Preview highlighted in the Preferences window

Chapter 4. The VSAM/QSAM Wizard

The Advanced COBOL Development technology preview provides new features and capabilities for the VSAM/QSAM Wizard.

Generation

To open the wizard, select **File > New > Project > COBOL Project**. Enter the COBOL Project name in the Wizard, and then select the **Dataset Application** template. In the next window, you can select either the **Program**, **Driver Program**, or **Copybook**. You can also indicate CICS or Batch, and whether or not the program takes input, gives output, or both.

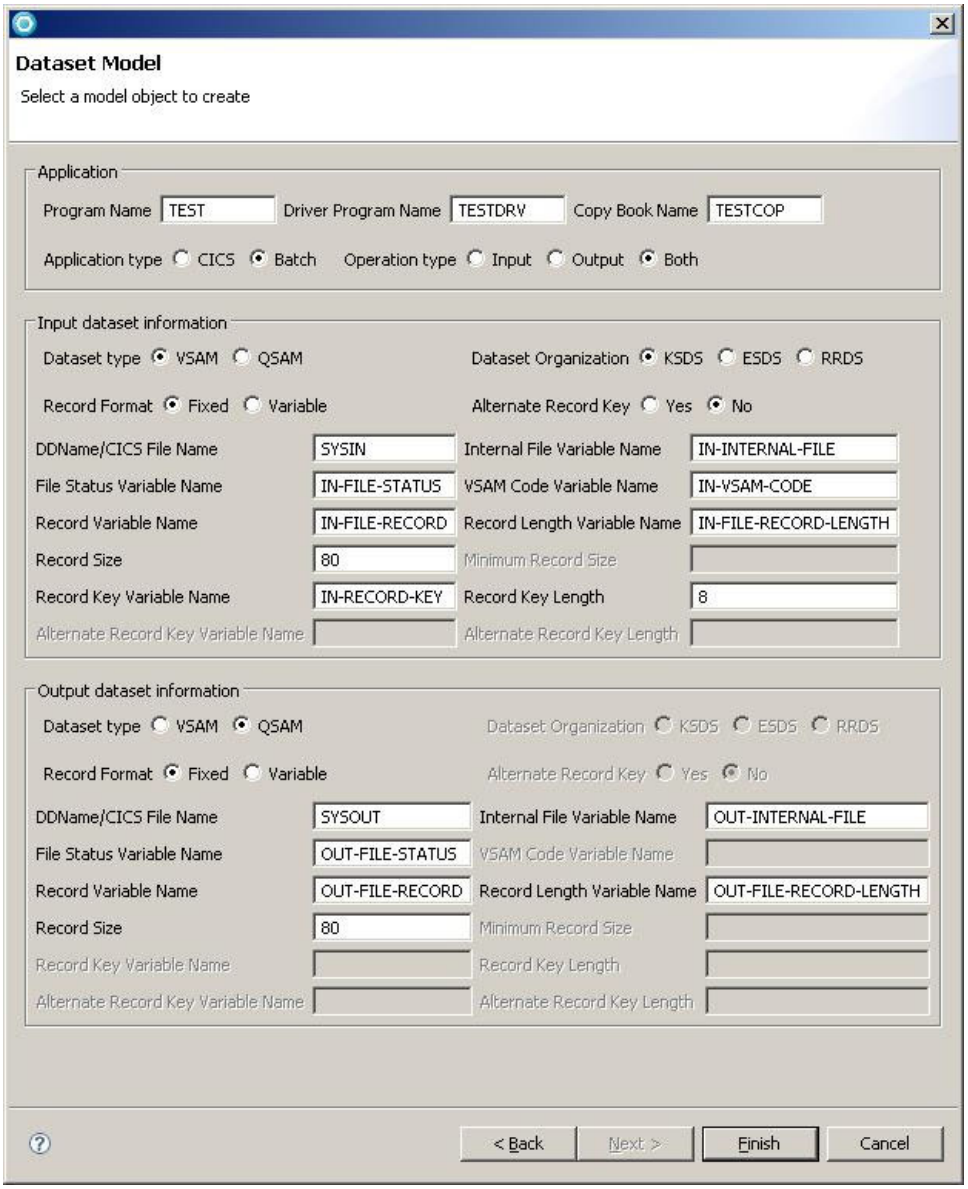

Figure 51. The dataset window

The CICS generated program allows only a VSAM dataset to be specified; only KSDS, ESDS, and RRDS are supported. QSAM is not supported. The generated CICS application allows both input and output against the specified data set.

When generating a program for batch, you can choose to have the program read input, write output, or both. You may also choose a different type of record for input or output. The options are KSDS, ESDS, RRDS, or QSAM. Table 3 shows how the types are accessed. See the *Enterprise COBOL Language Reference* for more information.

Table 3. Options for input and output

	QSAM	**KSDS**	**ESDS**	**RRDS**
Input	Organization is SEQUENTIAL; Access mode is SEQUENTIAL.	Organization is INDEXED; Access mode is DYNAMIC.	Organization is SEQUENTIAL; Access mode is SEQUENTIAL.	Organization is RELATIVE; Access mode is DYNAMIC.
Output	Organization is SEQUENTIAL; Access mode is SEQUENTIAL.	Organization is INDEXED; Access mode is RANDOM.	Organization is SEQUENTIAL; Access mode is SEQUENTIAL.	Organization is RELATIVE; Access mode is DYNAMIC.

Usage

The generated programs provide a good basis for developing applications that access VSAM and QSAM data sets; in nearly all cases some modification is necessary.

The generated batch applications have methods to open and close the data sets, read input, and write output. The application also accepts parameters. There is a driver program which calls the application with a parameter to copy the input data to the output dataset. When some settings are made, the driver is not created.

The CICS generation also creates an application and a sample driver program. The CICS program is designed to take parameters from an **EXEC** call. The program can also be modified to perform actions independently.

Limitations

The CICS project generated is not available from the **z/OS Projects View**. You can also use the Navigator view by selecting **Window > Show View > Navigator** from the menu bar.

Appendix. Notices

This information was developed for products and services offered in the U.S.A. IBM® might 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 might be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right might 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 might 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 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 might 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 might 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 might 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 Corporation
TL3B/062
3039 Cornwallis Road
RTP, NC 27709-2195
U.S.A.

Such information might be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information 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.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You might copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You might copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

If you are viewing this information softcopy, the photographs and color illustrations might not appear.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, other countries or both:

- IBM
- developerWorks
- Passport Advantage
- Rational
- Redbooks
- WebSphere

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names might be trademarks or service marks of others.

Printed in USA