Siebel Retail Finance Design Tools Guide

Version 2007.1

October 2007

ORACLE

Contents

1 What's New in This Release

What's New in Siebel Retail Finance Design Tools Guide, Version 2007.1

This guide has been updated to reflect the product version change. It was previously published as *Siebel Retail Finance Design Tools Guide*, Version 2007.

2 Design Tools Overview

This chapter provides an overview of Oracle's Siebel Retail Finance Design Tools, and contains the following topics:

- About the Class Builder
- About the Method Definer
- About the Validation Definer
- About the Session Builder
- About the Design Aids and Utilities

About the Class Builder

The Class Builder aids the design of Siebel Retail Finance standard entity beans. The tool makes use of already-existing attributes within the project to assist in class definition. The Class Builder also automatically updates background properties within the Rational Rose model. These help in the automatic code generation. You use the Class Builder from the Class Builder tab on the Design Tools Palette.

About the Method Definer

The Method Definer aids the design of Siebel Retail Finance standard entity and session beans. The tool updates the Siebel properties of the methods including the overview and behavior. The Method Definer makes it easier to create new methods, as it is easier to add parameters to the signature of a method and define the return types. You use the tool from the Class Builder or the Session Builder tab on the Design Tools Palette.

About the Validation Definer

The Validation Definer aids in the design of Siebel Retail Finance standard entity beans. The Validation Builder creates common validators for attributes that are held in the Rational Rose model (.mdl) file. These common validators are then available to all designers when creating the attributes of entity beans. This availability means that the designer can select an existing validator, which speeds up the process. The validators are held in a single class, which is generated by the code generator. You use the tool from the Validation Definer tab on the Design Tools Palette.

About the Session Builder

The Session Builder aids in the design of standard session beans. The Session Builder also automatically updates background properties within the Rational Rose model. These help in the automatic code generation. You use the tool from the Session Builder tab on the Design Tools Palette.

About the Design Aids and Utilities

The Design Aid and Utilities assist the job of the designers. Most of the utilities produce reports of the models. These allow the designers to get a quick view of the dependencies and associations within the model. They also assist in the development stage to identify all required methods and assist in development planning and provide a method of handing over the designs to the developers by generating both design documentation and code. You run all of the utilities from the Utilities tab on the Design Tools Palette. The utilities are summarized in the topics that follow.

About the Attribute Class Definer

The Attribute Class Definer makes sure that attributes defined for a particular class are made available for use throughout the model. When you run the Attribute Class Definer, the attributes package is updated with the details of all attributes defined in the model.

About the Design Documentation Builder

The Design Documentation Builder produces standard design documents. The documentation consists of project directories that contain financial objects, sessions, parameter objects, parameter object factories, external interfaces, and Java class documents.

About the Model Exporter

The Model Exporter exports XML from the Rational Rose designs. This XML is used by all other Siebel Retail Finance tools to develop the products. The Model Exporter generates a single XML file and outputs it to a chosen directory.

About the Model Validator

The Model Validator allows designers to check that their designs meet design standards. This is to make sure that the code can be generated quickly and easily from the models when the design phase is over. The tool performs class error checks, attribute error checks, function error checks, and process error checks on the model.

About the Model Comparison Tool

The Model Comparison Tool compares the differences between two models. The tool aids designers and developers in the following ways:

- To facilitate handover of updated Rational Rose models to the development team
- To help measure progress during the design phase
- To produce input to the review of design artifacts
- To help synchronization of streams within the design process

The tool produces reports in HTML format for each of the following:

- Banking objects
- Banking processes
- Parameter objects
- Common validators
- Validator lengths
- Constants

You use these comparison reports to identify changes to a model during the design, development, and testing phases of a project.

XML Syntax Exceptions

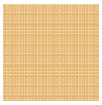
There are some fields in the Design Tools into which you cannot enter certain characters because these entries would result in an XML file that is not well formed when you export the model as an XML file. The characters you cannot enter are:

& < > “ ‘

If you try to enter any of these characters, a warning message is displayed.

The following lists shows, for the relevant tools, the fields in which you are not allowed to enter the characters:

- Class Builder Tool
  - Class Name on the Class Information tab
  - Table Name on the Class Information tab
  - Package Name on the Class Information tab
  - Attribute Name on the Attribute Definer tab
  - Table Column on the Attribute Definer tab
- Method Definer
  - Method Name
- Validation Definer
  - Name
  - Pattern
- Session Builder Tool
  - Session Name
  - Package Name
  - Request ID



Refactoring

If you change the name of a class, method, or attribute, you must update references to the name in the model. This updating is known as *refactoring.*

Changes that are made in the model when you refactor a name include the following:

- When you refactor a class name, the Attribute overview for all attributes on that class is updated accordingly.
- References to renamed attributes in the Overview and Behavior sections are updated across the model.
- FindBy method names for a renamed class are updated.
- The parameter lists and return values of all methods in the model are updated.
- The tree views in the Method Definer and Process Definer are updated.

When you click the Refactor button, a find and replace dialog is started. You can replace all occurrences of the text in the Find What field with the new refactored name; however, you must remember that this replacement could potentially cause inconsistencies if an object, attribute, or method of the same name already exists elsewhere in the model.

In the dialog there is a Match Case check box, which allows you to perform a case-sensitive search. There is also a Find Whole Word Only check; if you select this check box and try to replace, for example, the word “Actor”, words like "ActorGroup" are skipped.

See the following sections for more information about refactoring specific names:

- Refactoring a Class Name on page 16
- Refactoring an Attribute Name on page 21
- Refactoring a Method Name on page 29
- Refactoring a Validator Name on page 40

3 Using the Class Builder

This chapter describes how to use the Class Builder tool. When you have created a new model using the Siebel Retail Finance framework, you can design the entities. The Class Builder allows you to create new class packages in the Banking Objects (BO) section of the model. It allows you to create and delete Domain Layer packages and classes.

This chapter contains the following topics:

- Starting the Class Builder
- Defining Classes
- Defining Attributes
- Defining findBy Methods
- Defining Methods

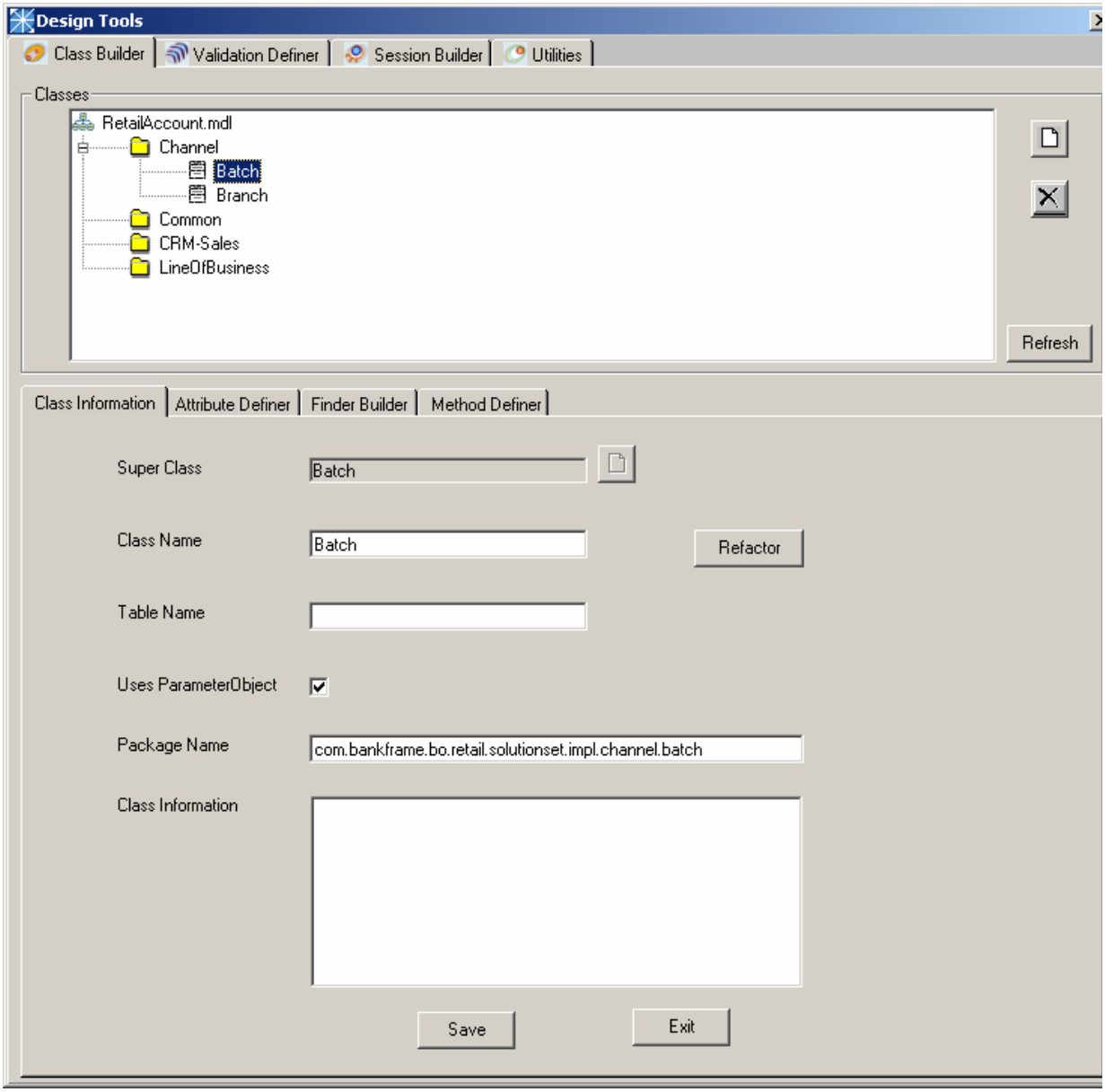
Starting the Class Builder

To start the Class Builder, navigate to Tools > Siebel in Rational Rose, and click the Class Builder tab in the Design Tools Palette; see Figure 1.

On loading, all of the existing classes are shown in the Classes tree. There is only one class per package, so the creation and maintenance of the packages are done in the background.

The elements in the Class Information section become active when you select an existing class or create a new class. Any classes that are contained within a write-protected package appear with a lock symbol next to them.

Figure 1. Class Builder



Defining Classes

You can add new classes, and amend or delete existing classes. To perform these tasks, first click the Class Information tab.

Creating a New Class

When you create a new class, you define the Banking Object (BO) grouping that the entity belongs to. A new class is added to the selected BO Grouping of the Rational Rose model.

The architecture is a four-layer object architecture, of which the lowest layers are the Module Layer and the Domain Layer. In the production release, a new class is added at the Module Layer and a new class with the same name is also added to the associated Domain Layer package. By default, a number of methods are added to the classes:

- The create, amend, and delete methods on the Module Layer class
- A findByPrimaryKey method on the Domain Layer class

In the delivery release, the new class is only added at the Domain Layer and all of the default methods are added to the class.

To create a new class

1. Click the  button next to the tree view.

   The Select Parent Package screen is displayed. This screen allows you to define the BO Grouping that you want to add the Entity to.

2. Select a package and click OK.

   The Class Information section becomes active at this point.

3. Enter a value in the Class Name field, and if required, in the Table Name, and Class Information fields.

4. Select the Uses ParameterObject check box, if you want the class to use a nonfunctional parameter object.

   Selecting this check box, creates a background property that indicates to the code generator that a parameter object containing all of the attributes of the class must be created. This property is used in the create () and amend () methods of the object. If you do not select this check box when you create the class, you can amend it afterwards and save it with the list of nonfunctional parameter objects being refreshed with the class name.

5. Click the button next to the Super Class field.

   A Select Super Class screen is displayed. In the production release, the list contains all of the Sector Core Layer Classes. In the delivery release you can select the superclass from either the Module Layer or Domain Layer.

6. Select a superclass and click OK.

7. Click Save.

   The class information is also saved if you select one of the other tabs, for example, FinderBuilder.

   When the entity is saved, the Package Name field is updated with the package name that you selected, plus the name of the entity. You can change the package name if required.

Amending a Class

You can amend all of the information for a class apart from the superclass.

To amend a class

1. Select a class in the Classes tree view.
2. Change the fields in the Class Information section as required.
3. Click Save.

Refactoring a Class Name

When you rename a class, you must make sure that all references to the class elsewhere in the model are changed accordingly. This process is known as refactoring; for more information, see Refactoring on page 12.

To refactor a class name

4. Select the class in the Classes tree view.
5. Type the required name in the Class Name field.
6. Click Refactor.
7. In the Find Replace dialog, replace occurrences of the old name with the new name as required.

   This replacement updates any references to this class name in the model.
8. Wait for the refactoring process to complete, and click Save.

Deleting a Class

In the production release, both the Module Layer and Domain Layer classes are deleted. However, in the delivery release, only the Domain Layer class is deleted.

To delete a class

1. Select a class in the Classes tree view.
2. Click the  button.
3. Click Yes to confirm the deletion.

   The selected class is deleted from the model.

Defining Attributes

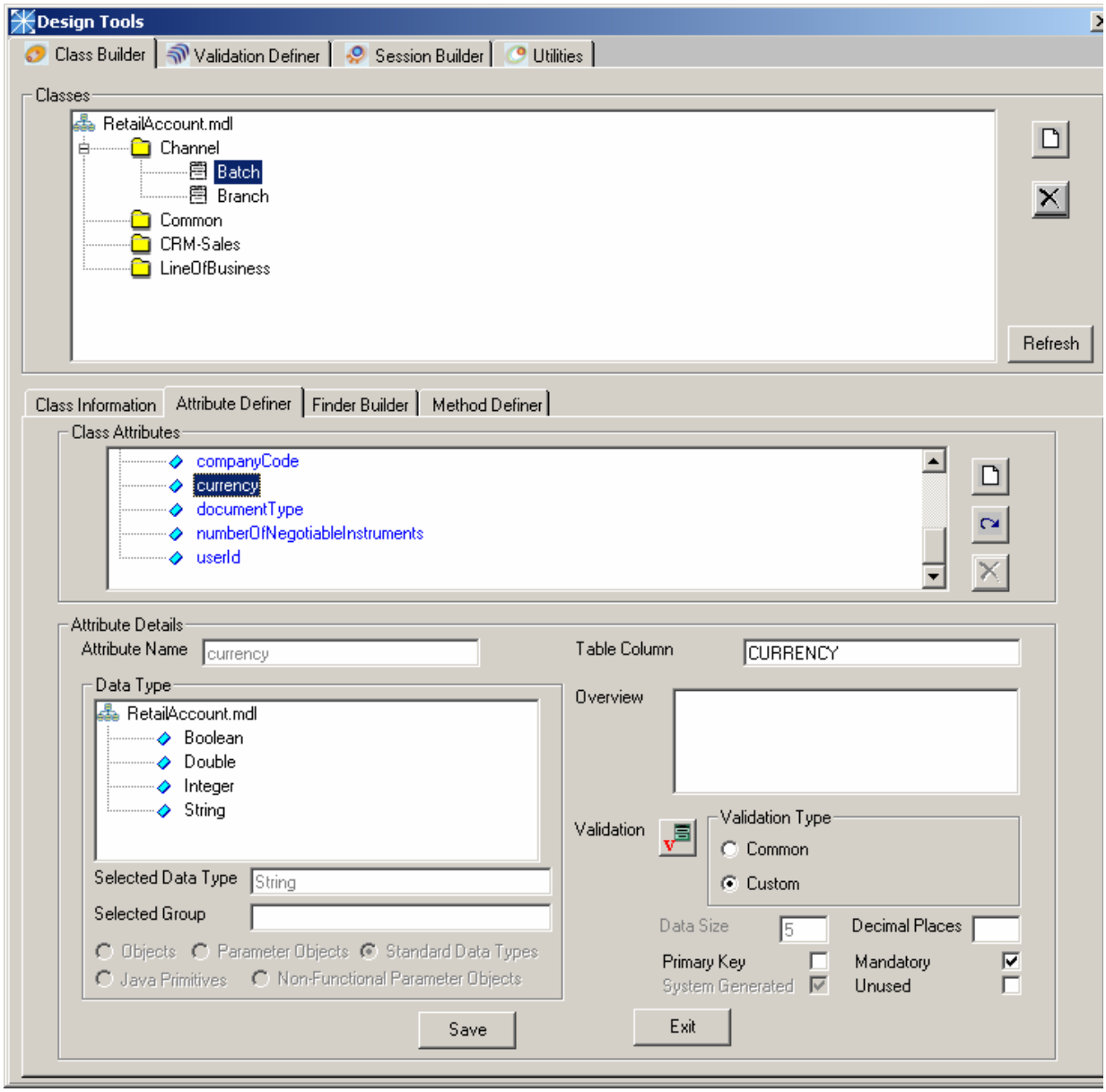
You can add new attributes, and amend or delete existing attributes. To perform these tasks, first click the Attribute Definer tab, as shown in Figure 2.

In the production release, the classes that are displayed are in the Module Layer, and the Class Attributes List only shows attributes of the Module Layer and its parent classes.

In the delivery release, the classes that are displayed are in the Domain Layer and the Class Attributes List shows attributes of the Domain Layer and its parent classes.

In the Class Attributes list, the attributes that are in the actual class are shown in black. The attributes of the parent classes are shown in blue.

Figure 2. The Attribute Definer



Adding New Attributes

You can create a new attribute by defining the details of a new attribute, or by basing the new attribute on an existing attribute. The latter approach allows you to add an existing application or system attribute to the entity. The advantage of this approach is that the attribute definition is complete, which makes the design quicker and the model more consistent.

The tool does not allow attributes of the same name to be defined on an entity.

To add a new attribute

1. Either click the  button to define a new attribute or click the  button to add an existing attribute.
2. If you clicked the  button, the Add Existing Attribute screen is displayed.
3. Select an attribute and click OK.
4. (Optional) If you clicked the  button, making the Attribute Details section active, complete the Attribute Details.

   The fields are described in the following table.

Field	Comments
Attribute Name	Type the name of the attribute.
Table Column	Type the name of the database column to which the attribute maps.
Overview	Type a description of the attribute.
Date Type	Select a data type for the attribute by clicking in the Data Type list. You can change the view in the list by clicking one of the radio buttons: Objects, Parameter Objects, Standard Data Types, Java Primitives, Non-Functional Parameter Objects.
Validation Type	Select the Common or Custom radio button, then click the Validation button, depending on whether you require a custom or common validator.
Data Size	Displays the attribute length. This field is populated when you define a validator for the attribute.
Decimal Places	For attributes with a data type of Double, type the number of decimal places.
Primary Key	Select this check box to define the attribute as a primary key. In this case,

Field	Comments
	the attribute has mandatory validation, and the System Generated check box is enabled.
Mandatory	Select this check box, if mandatory validation is required. By default, the attribute has optional validation.
System Generated	Click this check box for a primary key attribute that is to be system generated, that is, the attribute is taken from the system on the creation of an entity.
Unused	Select this check box, if the attribute is not to appear in the generated XML.

5 If you selected Common, displaying the Select Common Validation for Attribute screen, select a Common Validator from the list.

The details of the validator are displayed in the Overview section.

6 Click Save.

If the data type of the validator is not valid, you are prompted to select a validator with the correct data type to match the attribute data type. Make sure that the data types match as shown in the following table:

Attribute Type	Validator Type	Validator Value Type	Validator Length
String	String	Any one of the Validator value types	Validator Length required
Double	Number	N/A	Validator Length required
Boolean	Boolean	N/A	Validator Length required
Boolean	String	forLettersOrDigitsOnly	Validator Length required
Integer	String	forDigitsOnly	Validator Length required
Constant	String	Constant	Constant Key required
ByteArray	N/A	N/A	Not required

7 If you selected the Custom radio button, displaying the Enter Custom Validation for Attribute screen, complete the Overview details.

The fields are described in the following table.

Field	Comments
Name	Type the name of the attribute. The Name field defaults to validateAttributeName.
Type	Select the Validator Type. This type is either Boolean, Number, or String. The other fields in the Overview section are disabled depending on the Type chosen: ■ If you select Boolean, all other fields become disabled. ■ If you select Number, the Length, Max Length and Exact Length fields are enabled. ■ If you select String, the Value\Date, Value Type and Length fields are enabled.
Value\Date	Select a value as follows: ■ **Date Or Time.** You can then select Supply Pattern or System Date Pattern from the Value Type list. ■ **Value.** You can then select various values from the Value Type list.
Length	Select a validator length, or create a new length by clicking the button. You can also select Max Length or Exact Length, if required.
Value Type	Select a value depending on what you selected in the Value\Date field: ■ If you selected Date or Time in the Value\Date field: ■ **Supply Pattern.** The Pattern field becomes enabled. ■ **System Date Pattern.** The Pattern field defaults to System. In both cases, the Length, Max Length, and Exact Length fields are disabled. ■ If you selected Value in the Value\Date field: ■ Constant ■ forDigitsOnly ■ forLettersOnly ■ forLettersOrDigitsOnly, ■ forLettersOrDigitsOrWhiteSpacesOnly
Constants Key	Select an existing constants key from the list, or to add a new constant to the system, click the button beside the list. This button is only enabled when the attribute is a constant,

Field	Comments
	and you select Constant in the Value Type list.
Pattern	Type a specific date or time pattern, for example, yyyy:mm:dd.
Max Length	Select this check box if the attribute cannot exceed the defined attribute length.
Exact Length	Select this check box if the attribute cannot have a value larger or greater than the defined attribute length.

8 Click Save.

Amending an Attribute

If an attribute is from a superclass, you can amend the following fields: Attribute Name, Overview, and Data Type. All other parts of the Attribute Details section are editable. For an attribute in the actual class all fields are editable.

To amend an attribute

1 Select the attribute from the Class Attributes list.

2 Change the fields in the Attribute Details section.

3 Click Save.

Refactoring an Attribute Name

When you rename an attribute, you must make sure that all references to the attribute elsewhere in the model are changed accordingly. This process is known as refactoring; for more information, see Refactoring on page 12.

To refactor an attribute

1 Select the attribute in the Class Attributes tree view.

2 Type the required name in the Attribute Name field.

3 Click Refactor.

4 In the Find Replace dialog, replace occurrences of the old name with the new name as required.

This replacement updates any references to this attribute name in the model.

5 Wait for the refactoring process to complete, and click Save.

Deleting an Attribute

You cannot delete superclass attributes.

To delete an attribute

1. Select an attribute in the Class Attributes list.
2. Click the  button.
3. Click Yes to confirm the deletion.

   The selected attribute is deleted from the model.

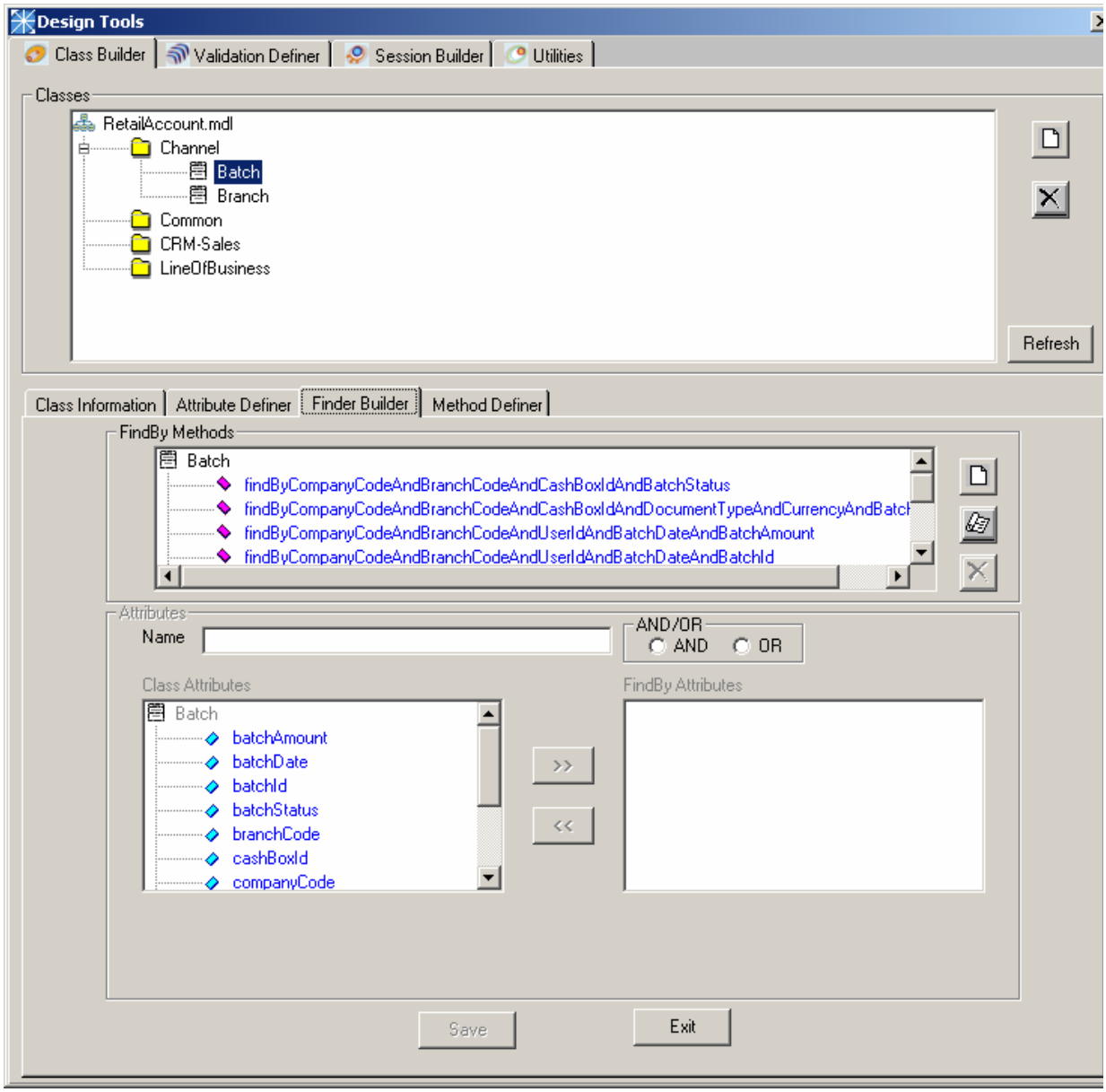
Defining findBy Methods

You can add findBy methods, and amend or delete existing methods. To perform these tasks, first click the Finder Builder tab, as shown in Figure 3.

The Finder Builder allows you to create, amend, and delete both simple and complex findBy methods.

A findBy method can be what is called an OR finder or an AND finder:

Figure 3. The Finder Builder



- An OR finder finds objects that fulfill any of the parameters passed in.
- An AND finder finds any objects that fulfill the combination of parameters.

Creating a Simple findBy Method

When you create a simple findBy method, you define whether the method is an AND finder or an OR finder, and you add the attributes to the method. Simple findBy methods are used if the attribute already exists on the Entity EJB.

To create a simple findBy method

1. Click the  button.
2. Select the AND radio button or the OR radio button, depending on whether the method is an AND or an OR finder.
3. Select an attribute from the Class Attributes list and click the >> button.

   To remove an attribute from the findBy method, select the attribute in the FindBy Attributes list and click the << button.
4. Click Save.

Creating a Complex findBy Method

When you create a complex findBy method, you define whether the method is an AND finder, or an OR finder, and you add the attributes to the method. In addition, you define rules for the attributes. Complex findBy methods are used if you want custom findBys, for example, if you want to change the name of the findBy or specify a new attribute.

To create a complex findBy method

1. Click the  button.
2. Type a name for the finder in the Name field.
3. Select the AND radio button or the OR radio button, depending on whether the method is an AND or an OR finder.
4. Select an attribute from the Class Attributes list and click the >> button. To remove an attribute from the findBy method, select the attribute in the FindBy Attributes list and click the << button.
5. To amend the rules on the parameters, double-click one of the attributes in the FindBy Attributes list.

   The Define FindBy Rules screen is displayed.
6. Select one of the following rules from the drop-down list, and click OK:
   - equal to =
   - less than <
   - less than or equal to <=
   - greater than >
   - greater than or equal to >=

- not equal to !=
- Range

7 Click Save.

Amending a findBy Method

In the production release, all of the Finder methods in the list are from the Module Layer and are editable.

In the delivery release, the Module Layer methods appear in blue and are not amendable. The Implementation Layer methods are shown in black and are amendable.

For all findBy methods, you can add or remove attributes. You can also change the finder method from an AND method to an OR method or vice versa. In the case of complex findBy methods, you can also amend the rules on each parameter.

To amend a findBy method

1 Select the method from the FindBy Methods list.

2 Change the details of the method, as required.

3 Click Save.

Deleting a findBy Method

You can delete a findBy method.

To delete a findBy method from a class

1 Select a method in the tree view.

2 Click the  button.

The selected method is deleted from the model.

Defining Methods for a Class

When you click the Method Definer tab on the Class Builder screen, the Method Definer starts. This allows you to define methods on the class. You define a method by giving it a name, a signature, and a return type. The Method Definer also allows you to define a more detailed behavior for a method. For information about the Method Definer, see the Method Definer chapter.

4 Using the Method Definer Tool

This chapter describes how to use the Method Definer tool. It contains the following topics:

- Starting the Method Definer Tool
- Defining Methods
- Defining the Method Signature
- Defining the Method Return

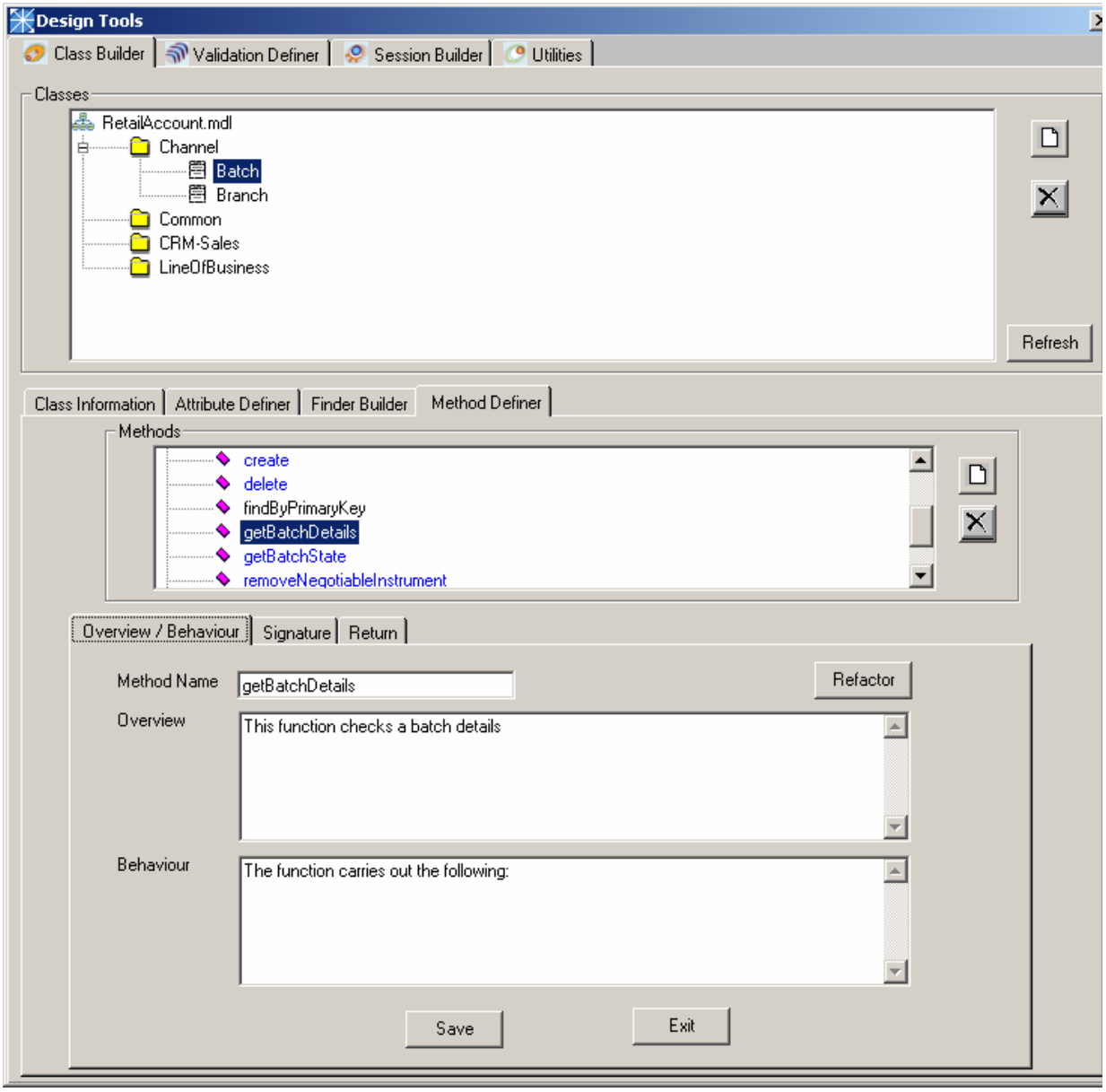
Starting the Method Definer Tool

You use the Method Definer from the Class Builder or the Session Builder; the functionality is the same in both cases. To start the Method Definer you click the Method Definer tab from the Class Builder, or the Process Definer tab from the Session Builder.

When you are using the Class Builder or Session Builder, and you select a class or session, the Method Definer tab and Process Builder tabs respectively become active, as shown in Figure 4.

NOTE: The text (Dep) is displayed before the method name of any method that is deprecated.

Figure 4. The Method Definer Tool



Defining Methods

You can create new methods, and amend or delete existing methods. To perform these tasks, first click the Overview / Behaviour tab.

Creating a New Method

When you create a new method, the Methods list is updated with the new method.

To create a new method

1. Click the  button in the Methods list.

   The Overview/Behavior section becomes active at this point.

2. Enter a name in the Method Name field, and if required, a method overview, and an overview of the method behavior in the relevant text fields.

3. Click Save.

Amending a Method

Methods of the actual class are shown in black, and methods of the superclasses are shown in blue. You cannot amend methods of a superclass.

To amend a method

1. Select the method in the Methods list.

2. Change the details

3. Click Save.

Deleting a Method

You can delete a method.

To delete a method

1. Select a method in the Methods list

2. Click the  button.

3. Click Yes to confirm the deletion.

   The selected method is deleted from the model.

Refactoring a Method Name

When you rename a method, you must make sure that all references to the method elsewhere in the model are changed accordingly. This process is known as *refactoring.*

You cannot rename inherited methods.

To refactor a method name

1. Select the method in the Methods list.

2. Type the required name in the Method Name field.

3 Click Refactor.

4 In the Find Replace dialog, replace occurrences of the old name with the new name as required.

This action updates any references to this method name in the model. The Business Process Diagram (BPD) is also renamed.

5 Wait for the refactoring process to complete, and click Save.

Defining the Method Signature

You can create, amend, and delete the parameters that constitute the signature of the selected method. To perform these tasks, first click the Signature tab, as shown in Figure 5.

Adding a Parameter

The Method Signature Definer allows you to select parameters for the selected method. You can select parameters from a number of lists comprising all the attributes, objects, parameter objects, primary key classes, and nonfunctional parameter objects that are in the project. You can also specify user-defined parameters. An example of such a parameter is a system-specific ID that is passed in from the front end but is not modelled as part of any entities.

To add a parameter from a list

1 Select one of the radio buttons: Attributes, Objects, Parameter Objects, Primary Key Classes and Non-Functional Parameter Objects.

2 Select from the Attributes list and click the >> button.

3 In the case of Objects, Parameter Objects, Primary Key Classes and Non-Functional Parameter Objects, you can click the >>> button.

This action adds a collection containing these objects to the Parameter lists. When you add a parameter object, or nonfunctional parameter object, the suffix “Impl” is added to the name of the parameter object.

4 Click Save.

To add a user-defined parameter

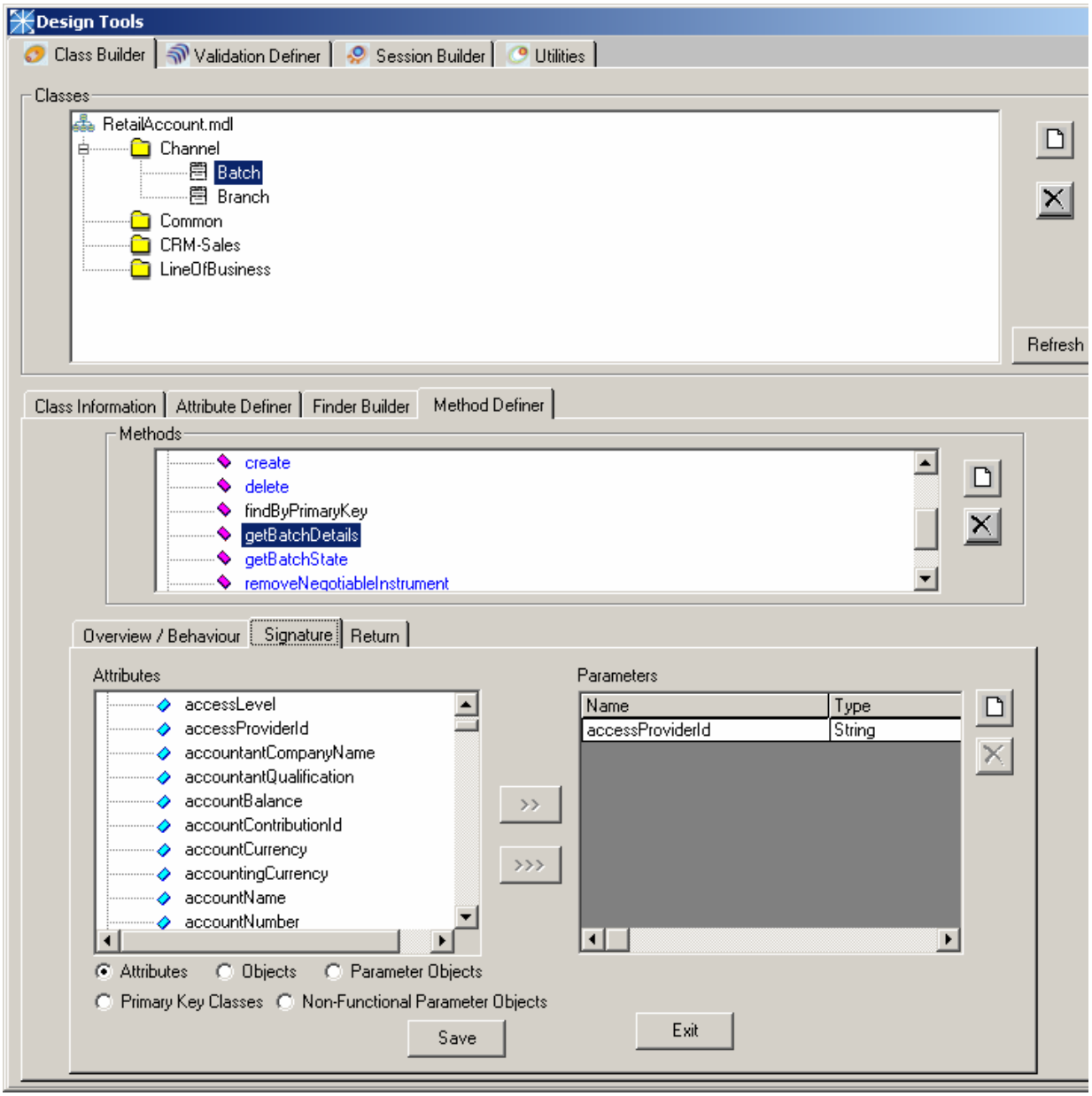
1 Click the  icon beside the Parameters list.

2 The Add Parameter screen is displayed.

3 Enter a name and data type for the parameter.

4 Click Save.

Figure 5. Method Definer; defining the method signature



Deleting a Parameter

You can delete a parameter.

To delete a parameter from the parameter list

1 Select the parameter from the Parameters list.

2 Click the  icon.

3 Click Yes to confirm the deletion.

The selected method is deleted from the model.

Defining the Method Return

You can define the primitives and objects that constitute the return of a selected method. To perform these tasks, first click the Return tab, as shown in Figure 6.

Adding a Return

The Method Return Definer allows you to select primitives and objects as returns for the selected method. There are a number of lists comprising all the objects, parameter objects, and primary key classes that are in the project. You can also define the return as Void.

To add a return

1 Click one of the radio buttons: Objects, Parameter Objects, and Primary Key Classes.

2 Select a data type from the list and click either the >> button to add a single selected item, or the >>> button to add a collection.

This action adds a collection such as a vector or enumeration containing these objects to the return.

3 Repeat steps 1 and 2 as required.

4 Click Save.

The Return Type field displays the type of object returned. If you specify only one return type, the name of that type is displayed, if you click the >>> button, Collection is displayed, and if you specify more than one return type, MultiTypeCollection is displayed.

Deleting a Primitive or Object from the Return List

You can delete a primitive or object from the return list.

To delete a primitive or object from the return list

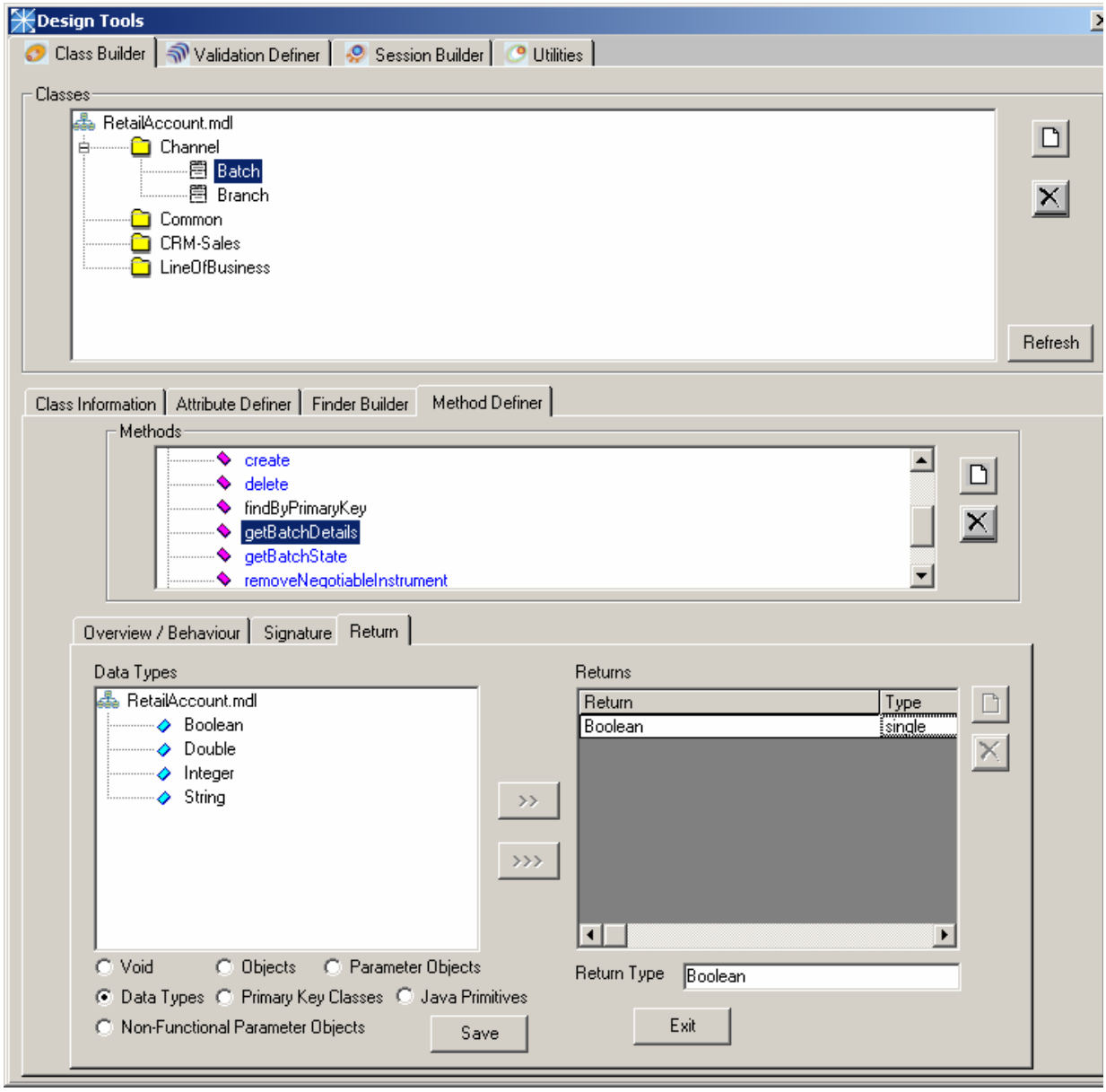
1 Select the primitive or object in the Returns list.

2 Click the  button.

3 Click Yes to confirm the deletion.

The selected primitive or object is deleted from the model.

Figure 6. Method Definer; defining the method return



5 Using the Validation Definer Tool

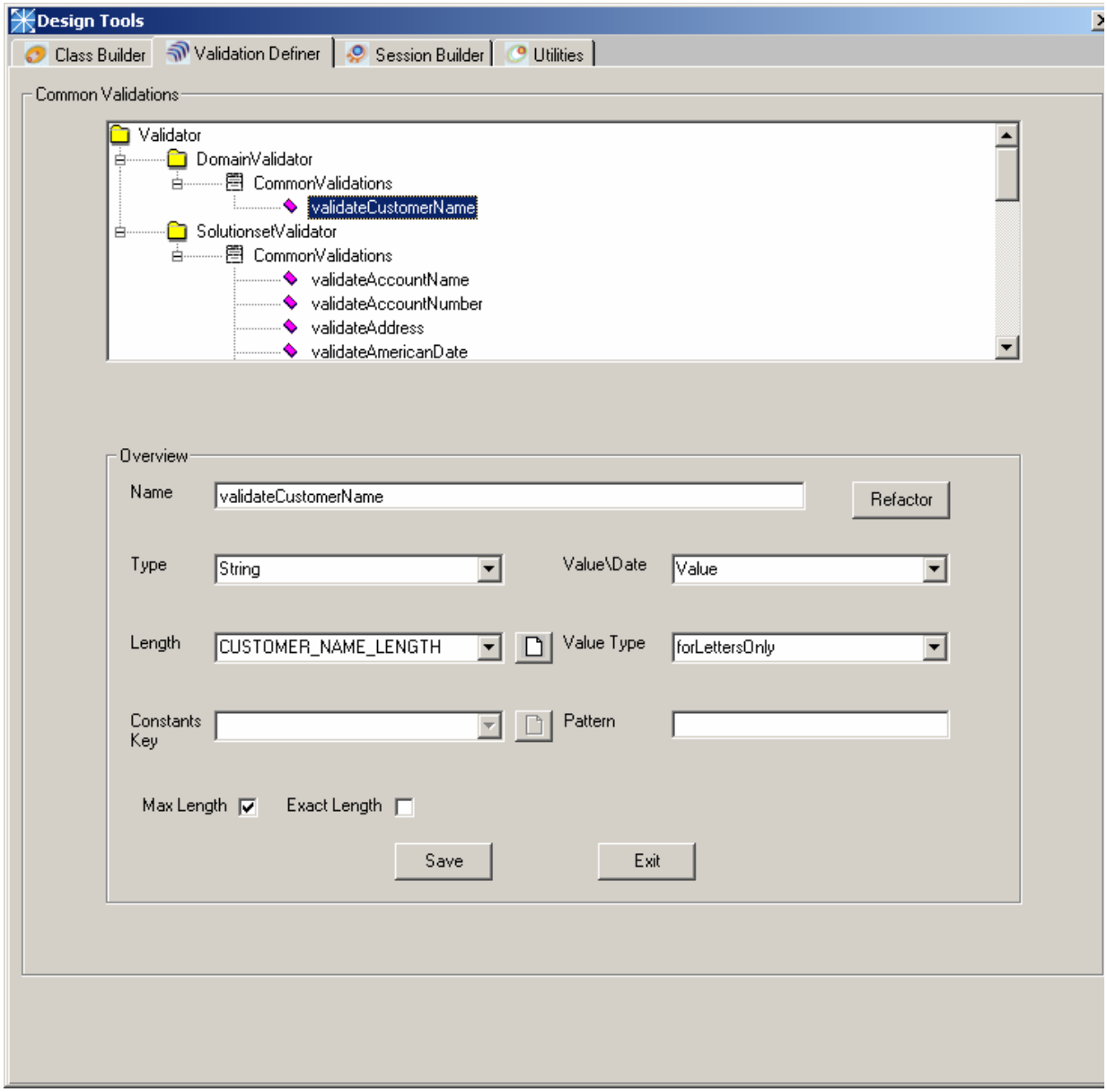
This chapter describes how to use the Validation Definer tool. It contains the following topics:

- Starting the Validation Definer
- Creating a Validation Group
- Creating a Validator
- Amending a Validator
- Renaming a Validation Group
- Deleting a Validation Group
- Deleting a Validator
- Refactoring a Validator Name
- Dragging and Dropping Validators and Validation Groups

Starting the Validation Definer

To start the Validation Definer, navigate to Tools > Siebel in Rational Rose, and click the Validations Definer tab in the Design Tools Palette, as shown in Figure 7.

Figure 7. The Validation Definer



The Validation Definer allows you to create new common validators in the model, and to delete and amend existing validators. Validators are contained within validation groups.

To view the properties of an existing validator, select it from the list. The details appear in the Overview section. The tree view displays all the Domain Layer and Solutionset Layer validators.

Creating a Validation Group

When you create a validation group, the description is displayed when you place the cursor over the validation group in the tree view. Within the new validation group, a class is automatically created,

which is merely a holder class for validators. The name of this class is <Group ID> appended with the text “Validations”.

To create a new validation group

1. In the Validator tree view, right-click DomainValidator or an existing group within the DomainValidator group.
2. Select New > Group.

   The Validation Group Definer screen is displayed.
3. Enter details in the Group ID and Description fields and click OK.

   The new validation group appears in the tree view.

   NOTE: You cannot create a validation group with the same name as an existing validation group at this hierarchy level in the tree view. An error message is displayed and you must enter the Group ID again.

Creating a Validator

You create a validator from a validation holder class. The validation holder class is created when you create a validation group.

To create a new validator

1. Right-click any of the validation holder classes within the DomainValidator group.
2. Select New > Validator.

   The Overview section is enabled.
3. Complete the Overview details.

   The fields are described in the following table.

Field	Comments
Name	Type the name of the validator. The Name field defaults to validateAttributeName.
Type	Select the Validator Type. This type is either Boolean, Number, or String. The other fields in the Overview section are disabled depending on the Type chosen: ■ If you select Boolean, all other fields become disabled. ■ If you select Number, the Length, Max Length, and Exact Length fields are enabled. ■ If you select String, the Value\Date, Value Type, and Length fields are enabled.

Field	Comments
Value\Date	Select a value as follows: ■ **Date Or Time.** You can then select Supply Pattern or System Date Pattern from the Value Type list. ■ **Value.** You can then select various values from the Value Type list.
Length	Select a validator length, or create a new length by clicking the button. You can also select Max Length or Exact Length if required.
Value Type	Select a value depending on what you selected in the Value\Date field: ■ If you selected Date or Time in the Value\Date field: ■ **Supply Pattern.** The Pattern field becomes enabled. ■ **System Date Pattern.** The Pattern field defaults to System. In both cases, the Length, Max Length, and Exact Length fields are disabled. ■ If you selected Value in the Value\Date field: ■ Constant ■ forDigitsOnly ■ forLettersOnly ■ forLettersOrDigitsOnly, ■ forLettersOrDigitsOrWhiteSpacesOnly
Constants Key	Select an existing constants key from the list, or to add a new constant to the system, click the button beside the list. This button is only enabled when the attribute is a constant, and you select Constant in the Value Type list.
Pattern	Type a specific date or time pattern, for example, yyyy:mm:dd.
Max Length	Select this check box if the attribute cannot exceed the defined attribute length.
Exact Length	Select this check box if the attribute cannot have a value larger or greater than the defined attribute length.

4 Click Save.

Amending a Validator

Only the Domain Layer validators are amendable.

To amend a common validator

1. Select the validator from the Common Validations tree view.
2. Change the fields in the Overview section as required.
3. Click Save.

Renaming a Validation Group

You can rename validation groups.

To rename a validation group

1. Right-click the validation group within the DomainValidator group.
2. Select Rename Group.

   The selected validation group name is now editable.
3. Enter the new name for the validation group.

Deleting a Validation Group

When you delete a validation group, the selected validation group, subgroups and validators are removed from the model.

CAUTION: You must take care when deleting existing validation groups as entity attributes can already reference validators within this validation group or subgroups.

To delete a validation group

1. Right-click the validation group within the DomainValidator group.
2. Select Delete > Group.

Deleting a Validator

CAUTION: You must take care in deleting existing validators as entity attributes can already reference these validators.

To delete a validator

1. Right-click the validator within the DomainValidator group.

2 Select Delete > Validator.

The selected validator is then removed from the model.

Refactoring a Validator Name

When you rename a validator, you must make sure that all references to the validator elsewhere in the model are changed accordingly. This process is known as *refactoring*. For more information, see Refactoring on page 12.

To refactor an existing validator name

1 Select the validator from the tree view and update the Name field in the Overview section.

2 Click Refactor.

3 Click Save when the refactoring process has finished.

This action updates any references to this validator in the model.

Dragging and Dropping Validators and Validation Groups

You can drag and drop all Domain Layer validation groups, validators, and validation holder classes. You cannot do this within the Solutionset Layer. You can move:

- A validation group into another validation group, if that validation group does not already contain a validation group with the same name as the validation group being dragged.
- A validation holder class into another validation group, if the validation group does not already contain a holder class with this name. Any validators defined within the holder class are also moved.
- A validator to another holder class, if the holder class does not already contain a validator with the same name.

6 Using the Session Builder Tool

This chapter describes how to use the Session Builder tool. When you have created a new model using the Framework, you can design the session beans. The Session Builder allows you to create new class packages in the Banking Processes section of the model. It allows you to create and delete Domain Layer packages and classes.

The chapter contains the following topics:

- Starting the Session Builder Tool
- Defining Sessions
- Defining Processes

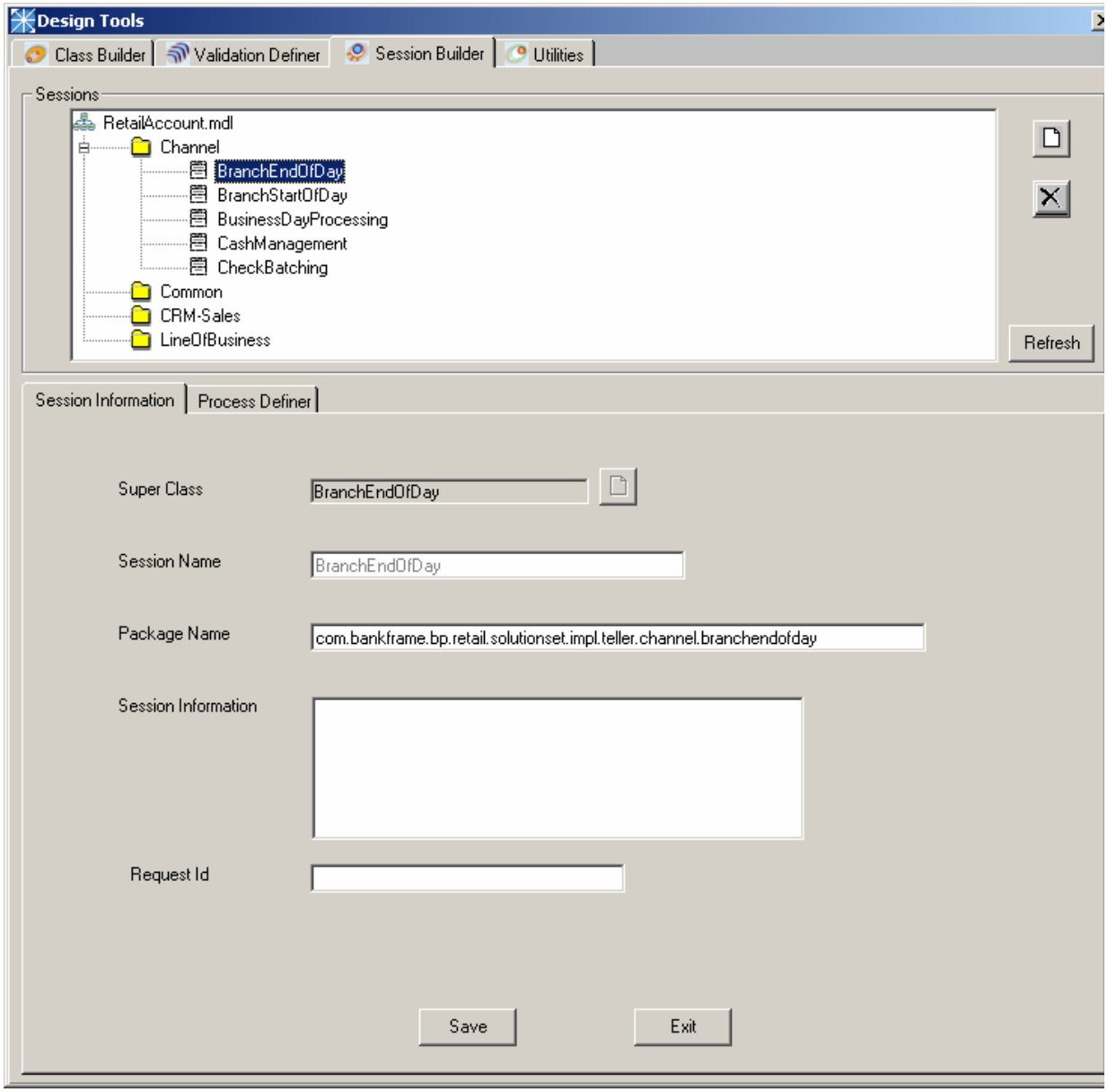
Starting the Session Builder Tool

To start the Session Builder, navigate to Tools > Siebel in Rational Rose, and click the Session Builder tab in the Design Tools Palette, as shown in Figure 8.

On loading, all of the existing sessions are shown in the Sessions tree. There is only one class per package, so the creation and maintenance of the packages occur in the background.

The elements in the Session Information section become active when you select an existing class or create a new class. Any classes that are contained within a write-protected package appear with a lock symbol next to them.

Figure 8. The Session Builder



Defining Sessions

You can create new sessions, and amend and delete new sessions. To perform these tasks, first click the Session Information tab.

Creating a New Session

When you create a new session, you define the Banking Process (BP) grouping that the session belongs to. A new class is added to the selected BP Grouping of the Rational Rose model.

The solution works within a two-layer session architecture. In the production release, a new session is added at the Module Layer and a new session with the same name is also added to the associated Domain Layer package. In the delivery release, the new session is only added at the Domain Layer.

To create a new session

1. Click the  button beside the Sessions tree view.

   The Select Parent Package screen is displayed. This screen allows you to define the BP Grouping to which you want to add the session.

2. Select a package and click OK.

   The Session Information section becomes active at this point.

3. Enter a name in the Session Name field, and if required, type a description in the Session Information field.

4. If required, enter a value in the Request Id field; this is output in the XML, and can be used later to populate the Routes table.

5. To save the information, click Save.

   The information is also saved if you select one of the other tabs, for example, Process Builder.

   When the session is saved, the Package Name field is updated with the package name that you selected, plus the name of the session. You can change the package name if required.

Amending a Session

You can amend all the fields in the Session Information, apart from the Name field.

To amend a session

1. Select a session in the tree view.
2. Change the fields in the Session Information section as required.
3. Click Save.

Deleting a Session

In the production release, both the Module and Domain Layer sessions are deleted. However, in the delivery release only, the Domain Layer session is deleted.

To delete a session

1. Select a session in the tree view.
2. Click the  button.
3. Click Yes to confirm the deletion.

   The selected session is deleted from the model.

Defining Processes

When you click the Process Definer tab on the Session Builder screen, the Method Definer starts. This Method Definer allows you to define processes on the session. You define a process by giving it a name, a signature, and a return type. The Method Definer also allows you to define a more detailed behavior for a process. For information about the Method Definer, see the Method Definer chapter.

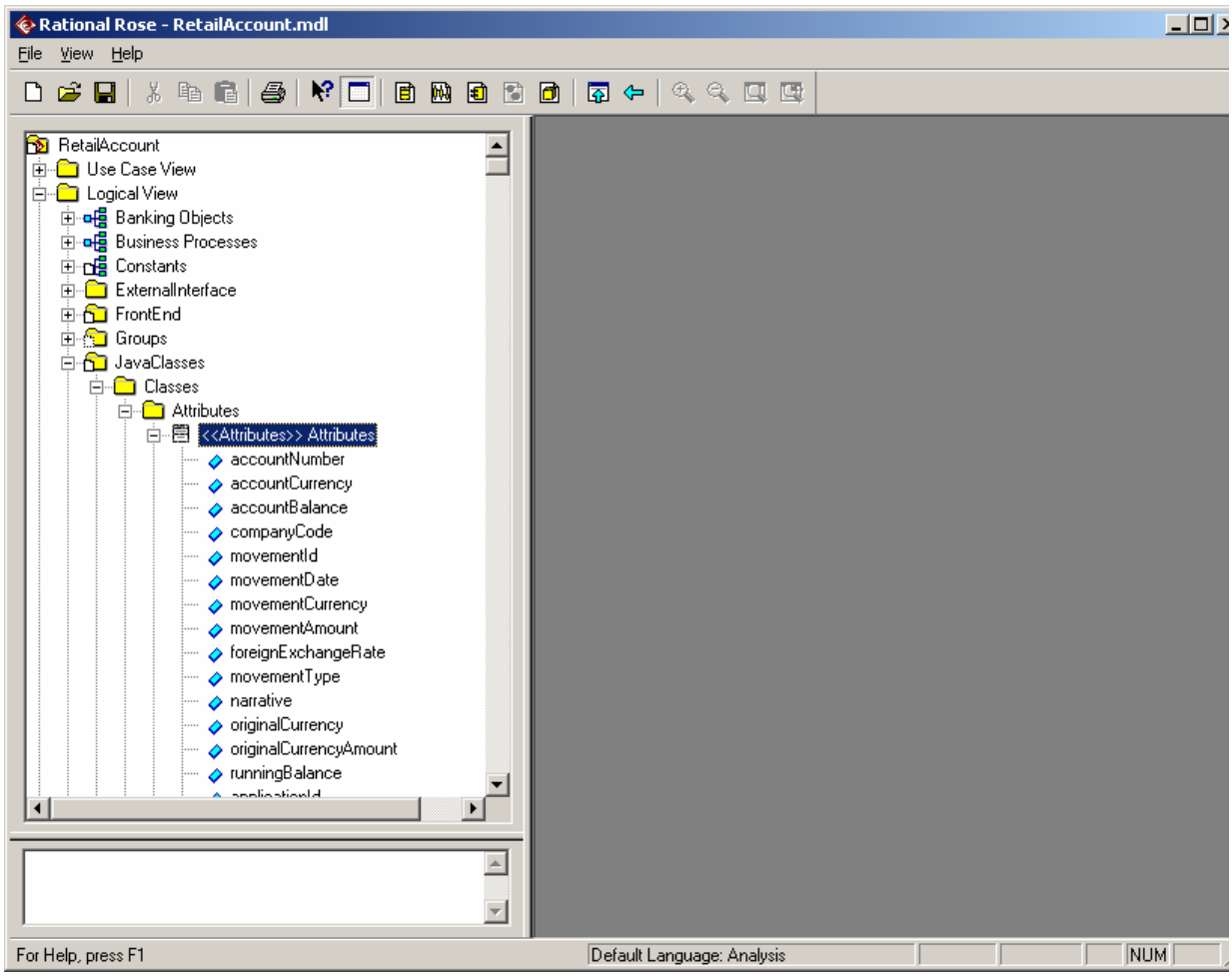
7 Using the Attribute Class Definer

The Attribute Class Definer makes sure that attributes defined for a particular class are made available for use throughout the model.

To run the Attribute Class Definer, click its icon on the Utilities tab of the Design Tools Palette.

When you run the Attribute Class Definer, the *Attributes* package in Rational Rose is updated with the details of all attributes defined in the model, as shown in Figure 9.

Figure 9. The Attribute Class Definer



8 Using the Design Documentation Builder

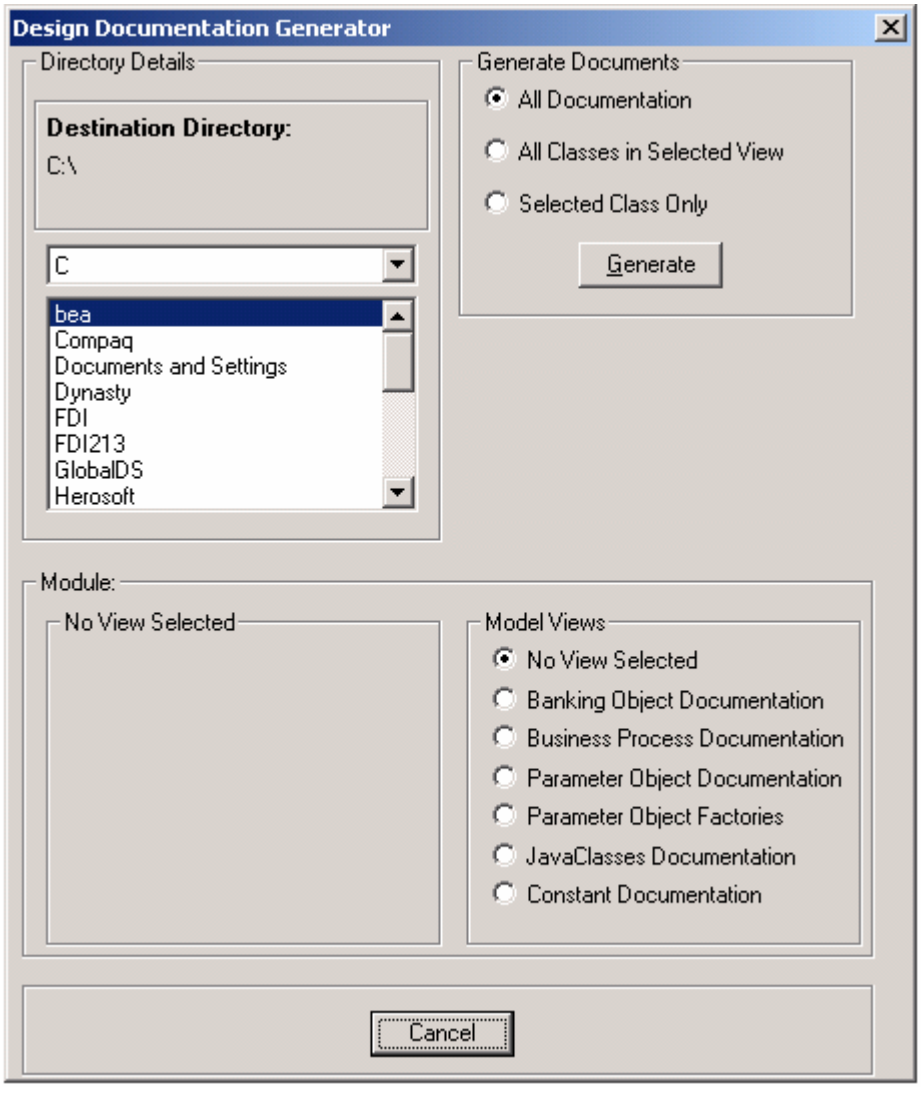
When you have completed the design phase for a requirement, you can use the Design Documentation Builder to create standard design documents. You can generate design documentation for the whole model, for a selected model view, or for a single class. This chapter contains the following topics:

- Generating Design Documentation
- The Design Documentation that is Generated

Generating Design Documentation

When you click the Generate Design Documents icon on the Design Tools Palette, the Design Documentation Generator is displayed as shown in Figure 10.

Figure 10. The Design Documentation Generator



To generate design documentation

1. Click Generate Design Documents on the Design Tools Palette.
2. Select the Destination Directory for the documents using the drop-down list and list box.
3. Click one of the following radio buttons according to the documentation you want to generate:
   - All Documentation
   - All Classes in Selected View
   - Selected Class Only
4. If you clicked All Classes in Selected View, click a radio button under Model Views corresponding to the selection of design documents that you require:
   - Banking Object Documentation
   - Business Process Documentation
   - Parameter Object Documentation
   - Parameter Object Factories
   - JavaClasses Documentation
   - Constant Documentation
5. If you clicked Selected Class Only, make sure that one of the Model Views buttons is selected, and select a class from the list of classes displayed in the Module list.
6. Click Generate.

NOTE: There is a Windows restriction of 256 characters for a fully-qualified filename. Overloading operations with up to 10 parameters causes a maximum of 55 characters to be added to the name of the operation design file; it can be more for operations with more that 10 parameters. If the fully-qualified file name of the operation exceeds 256 characters, an error message is displayed with the message, `Path not found`, and the design document generation exits at this point. To overcome this problem, try generating the documents in a folder off the c:\ or d:\ root drive, or shorten the class or operation names.

The Design Documentation that is Generated

The following folders are created at the location you specify using the tool:

- Constants
- ExternalInterfaces
- FinancialObjects (banking object documentation)
- ParameterObjectFactories
- ParameterObjects
- JavaClasses

- Sessions (banking process documentation)

For each of these folders there is an index.html file that provides links to documents for all of the objects in that folder. There is also a top-level index.html file that provides links to the index.html file for each folder.

9 Using the Model Exporter Tool

This chapter describes how to use the Model Exporter tool, which you use to export your design model as an XML file. The chapter contains the following topics:

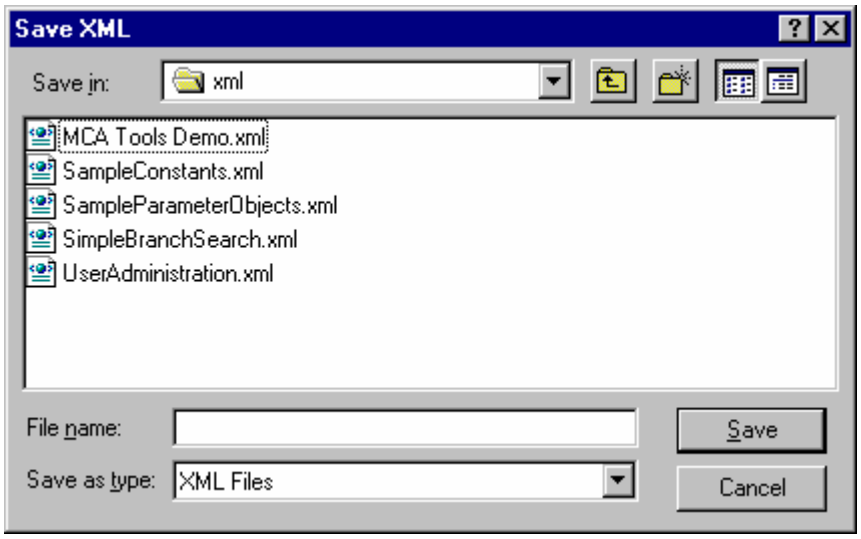
- Exporting the Model

Exporting the Model

You use the Model Exporter when a model is completed, or a module is taken from the repository. The tool exports the model as an XML file.

When you click the Export Model as XML icon on the Design Tools Palette, Figure 11 is displayed:

Figure 11. The Model Exporter Tool



To export the model

1. Click the Model Exporter icon on the Design Tools Palette.

   The Save XML screen is displayed.

2. In the Save in field list, select the location where you want to save the generated XML file.
3. Enter the name of the XML file you want to generate, or select an existing file to overwrite.
4. Click Save to generate the file.

10 Using the Model Validator Tool

This chapter describes how to use the Model Validator tool. It contains the following topics:

- The Validation Properties File
- Validations Performed by the Model Validator
- Validating the Model

The Validation Properties File

When you run the Model Validator tool, it reads a properties file to determine which validations it uses to validate the model, and whether a custom Rational Rose script is used. The following excerpt illustrates the format of the validation properties file:

```
# Custom Rose Script

customModelValidations.ebx

# Attribute Validations

AttributeMandatoryValidation

AttributeDataSize

# AttributeDataSizeNumeric

AttributeValidatorMethodDefined
```

The name of any custom Rational Rose script is specified in the first uncommented line of the properties file.

You can edit the validations properties file to determine which validations are used. To remove a validation, insert the # symbol at the beginning of the line.

Validations Performed by the Model Validator

Table 1 lists the standard validations performed by the Model Validator.

Table 1. Standard Validations

Section	Validation Description	Validation Code
General Errors	A check that the documentation field in the BankingObject Category is not blank.	GeneralBankingObjectCategoryDocumentation

Section	Validation Description	Validation Code
	A check that the BankingObject category contains a Domain Layer.	GeneralBankingObjectCategoryDomain
	A check that the BankingObject category contains a Module Layer.	GeneralBankingObjectCategoryModule
	The documentation field in the Domain Layer packages (stereotype DomainPackageObject) must contain part of the Java namespace.	GeneralBankingObjectDomainLayerCategoriesDocumentation
	The documentation field in the Module Layer packages (stereotype ModulePackageObject) must contain part of the Java namespace.	GeneralBankingObjectModuleLayerCategoriesDocumentation
	The BusinessProcess category documentation must contain com.bankframe.bp.	GeneralBusinessProcessCategoryDocumentation
	The BusinessProcess category must contain a Domain Layer.	GeneralBusinessProcessCategoryDomain
	The BusinessProcess category must contain a Module Layer.	GeneralBusinessProcessCategoryModule
	A check that the Domain Layer Categories contain qualified name documentation.	GeneralBusinessProcessDomainLayerCategoriesDocumentation
	A check that the Module Layer Categories contain qualified name documentation.	GeneralBusinessProcessModuleLayerCategoriesDocumentation
Validator Errors	A check that the model contains a Validator category.	ValidatorCatagoryExists
	The Validator category documentation must contain com.bankframe.validator	ValidatorCatagoryDocumentation

Section	Validation Description	Validation Code
	The Validator category must contain a Domain Layer.	ValidatorCatagoryDomainExists
	The Domain Layer Validator category must contain at least one class.	ValidatorDomainClasses
	The Validator category must contain a Module Layer.	ValidatorCatagoryModuleExists
	The Module Layer Validator category must contain qualified name documentation.	ValidatorCatagoryModuleDocumentation
	The Module Layer Validator category must contain at least one class.	ValidatorModuleClasses
	The class Common Validations must have a stereotype of <<DomainValidator>>, if it belongs to the DomainValidator package.	ValidatorDomainStereotype
	The class Common Validations must have a stereotype of <<SolutionsetValidator>>, if it belongs to the SolutionsetValidator package.	ValidatorModuleStereotype
Constant Class Errors	The model must contain a Constants category.	ConstantsCatagoryExists
	The Constants category documentation must contain com.bankframe.co.	ConstantsCatagoryDocumentation
	The Constants category does must contain a Constants class.	ConstantsClassExists
	The Constants class must have a stereotype of <<Constants>>.	ConstantsClassStereotype

Section	Validation Description	Validation Code
Attribute Class Errors	The model must contain an Attributes category.	AttributesCategoryExists
	The Attributes category must contain an Attributes class.	AttributesCategoryClasses
	The Attributes class must belong to the Attributes package.	AttributesClassStereotype
Class Errors	A check that no object starts with a small letter.	classstartslowercaseCheck
	A check that no object is misspelled or contains illegal characters.	classillegalcharacterCheck
	A check that no object inherits from multiple classes.	classMultipleInheritanceCheck
	A check that no object has duplicate associations.	classDuplicateAssocationCheck
	A check that no Solutionset objects exist without corresponding Domain Package Objects.	classdomainlayerclassCheck
	A check that every package name matches its session name.	classdifferentpackagenamCheck
	A check that findByPrimaryKey is correctly spelled.	classFindByPKCheck
	A check that the stereotype is correct in all cases.	classSterotypeCheck
	A check that every entity has at least one primary key attribute defined for it.	classPrimaryKeyCheck
	A check that every entity has at least one attribute.	classNoAttributesCheck
Attribute Errors	A check that attribute data size is defined.	attributeDataSizeCheck

Section	Validation Description	Validation Code
	A check that attribute table column name is defined.	attributreColumnCheck
	A check that attribute data type is defined.	attributeDataTypeCheck
	A check that attribute validation has been defined (stereotype).	attributeMandatoryValidationCheck
	A check that no attribute validation has been defined (nonstereotype).	attributeValidatorCheck
	A check that no attribute starts with a capital letter.	attributeLowercaseCheck
	A check that no attribute is misspelled or contains illegal characters.	attributeIllegalCharacterCheck
	A check that no object has duplicate attributes.	classDuplicateCheck
	A check that attribute data size is defined with a numeric value.	attributeDataNumericCheck
Function Errors	A check that every function overview has been completed.	functionOverviewCheck
	A check that every function behavior has been completed.	functionBehaviourCheck
	A check that every function's parameters have been properly defined.	functionParameterCheck
	A check that no function starts with a capital letter.	functionLowercaseCheck
	A check that no function is misspelled or contains illegal characters.	functionNameIllegalCharacterCheck
	A check that no function behavior contains illegal characters.	functionBehaviourIllegalCharacterCheck
	A check that no function overview contains illegal	functionOverviewIllegalCharacterCheck

Section	Validation Description	Validation Code
	characters.	
	A check to make sure that all the parameters for a findBy are attributes of the object the findBy is on.	functionFindByCheck
	A check that the function has a return type specified	functionReturnTypeCheck
	A check that any method that overwrites an existing method has the same return type as the original method.	functionReturnOverwriteCheck
	A check that any method that overwrites an existing method has the same return type as the original method.	functionReturnOverwriteCheck
Session Errors - Module Layer	A check that every session name does not contain an illegal character.	SessionModuleIllegalCharacter
	A check that the session name starts with a lower case letter.	SessionModuleStartsLowerCase
	A check that the session has a corresponding class in the Domain Layer.	SessionModuleDomainLayer
	A check that the session Package Name does not differ from the Class name.	SessionModuleDifferentPackageName
	A check that the session has the correct stereotype.	SessionModuleStereotype
	A check that every session name does not contain an illegal character.	SessionModuleIllegalCharacter
	A check that the session contains at least one operation.	SessionModuleOperationsExist
	A check that no method is overwriting a method with	SessionModuleMethodOverwritingDifferentReturn

Section	Validation Description	Validation Code
	a different return type.	
BPD Errors – Module Layer	A check that no BPD contains Module Layer objects.	BPDModuleNonDomainLayerObject
	A check that no BPD contains undefined classes.	BPDModuleUndefinedObject
	A check that the BPD has a process associated with it on the session.	BPDModuleMethodAssociated
	A check that the BPD does not have an undefined message.	BPDModuleUndefinedMessage
Process Errors - Module Layer	A check that the process is not named with an illegal character.	ProcessModuleNameIllegalCharacter
	A check that every process overview has been completed.	ProcessModuleOverviewDefined
	A check that every process overview does not contain illegal characters.	ProcessModuleOverviewIllegalCharacter
	A check that every process behavior has been completed.	ProcessModuleBehaviourDefined
	A check that every process behavior does not contain illegal characters.	ProcessModuleBehaviourIllegalCharacter
	A check that every process response has been completed.	ProcessModuleReturnDefined
	A check that no function starts with a capital.	ProcessModuleStartsLowerCase
	A check that all method arguments have a type.	ProcessModuleParameterValidType
Session Errors - Domain	A check that every session name does not contain an illegal character.	SessionDomainIllegalCharacter

Section	Validation Description	Validation Code
Layer		
	A check that the session name starts with a lower case letter.	SessionDomainStartsLowerCase
	A check that the session is not inherited from multiple objects.	SessionDomainInheritedMultiple
	A check that the session Package Name does not differ from the class name.	SessionDomainDifferentPackageName
	A check that the session has the correct stereotype.	SessionDomainStereotype
	A check that the session contains at least one operation.	SessionDomainOperationsExist
	A check that no method is overwriting a method with a different return type.	SessionDomainMethodOverwritingDifferentReturn
BPD Errors – Domain Layer	A check that no BPD contains Module Layer objects.	BPDDomainNonDomainLayerObject
	A check that no BPD contains undefined classes.	BPDDomainUndefinedObject
	A check that the BPD has a process associated with it on the session.	BPDDomainMethodAssociated
	A check that the BPD does not have an undefined message.	BPDDomainUndefinedMessage
Process Errors - Domain Layer	A check that the process is not named with illegal characters.	ProcessDomainNameIllegalCharacter
	A check that every process overview has been completed.	ProcessDomainOverviewDefined
	A check that every process overview does not contain	ProcessDomainOverviewIllegalCharacter

Section	Validation Description	Validation Code
	illegal characters.	
	A check that every process behavior has been completed.	ProcessDomainBehaviourDefined
	A check that every process behavior does not contain illegal characters.	ProcessDomainBehaviourIllegalCharacter
	A check that every process response has been completed.	ProcessDomainReturnDefined
	A check that no function starts with a capital.	ProcessDomainStartsLowerCase
	A check that all method arguments have a type.	ProcessDomainParameterValidType

Validating the Model

You validate a model by clicking one of the following icons on the Utilities tab on the Design Tools Palette:

- **Model Validator – Default.** Validates the model using the default properties file.
- **Model Validator – Select File.** Validates the model using a specified properties file.
- **Model Validator – Extended.** Validates the model using a custom Rational Rose script and associated properties file.

In each case, the tool performs validations according to the specified properties files and produces an HTML report file.

Running the Model Validator with the Default Properties File

The default properties file is the file that was last used by the Model Validator. However, if you are running the Model Validator for the first time, you are prompted to specify a properties file.

To run the Model Validator with the default properties file

1. Click Model Validator – Default.
2. Specify a properties file, if this is the first time you are running the tool.
3. Specify the location of the results file.

Running the Model Validator with a Selected Properties File

You can predefine multiple properties files, according to the requirements of different projects.

The file that you specify then becomes the default properties file.

To run the Model Validator with a selected properties file

1. Click Model Validator – Select File.
2. Select a properties file.
3. Specify the location of the results file.

Running the Model Validator with a Custom Rational Rose Script

To run the Model Validator with a custom Rational Rose Script, you must first create the script and a validation properties file in the same format as the sample files (sampleCustomValidations.properties and sampleCustomValidations.ebs) that are installed in the Design Tools installation directory.

To prepare the custom script and properties file

1. Extend the sampleCustomeValidations.ebs script, or create a new .ebs file to include the custom validations that you require. If you create a new .ebs file, make sure it is in accordance with the following specification:
   - All validation methods must be declared as Global at the start of the file
   - The code must implement the following three PUBLIC subroutines which are called from the Design Tools:
     - main
     - setValidationToCheck(validatorMethodName As String)
     - startValidationProcess
   - In the main subroutine method, the validation methods are initialized
   - In the setValidationToCheck subroutine, each validation check to be carried out is set. Only the validation names contained in the .properties file without the # are set
   - The startValidationProcess subroutine starts the validation on the model
2. Edit the sampleCustomValidations.properties file, and make sure that it contains the name of the Rational Rose script and a list of the validations to perform. For example:

```
# Custom Validations

#define EBX file to run below

sampleCustomValidations.ebx

#define validations to check below any that have # before them will be ignored

GeneralBankingObjectCategoryDocumentation
```

```
#GeneralBankingObjectCategoryDomain
```

3 Compile the ebs file into an ebx file.

4 Place both the .properties and .ebx file in the Design Tools installation directory where the Design Tools specific .ebx files are located.

To run the Model Validator with a custom script

1 Click Model Validator – Extended.

2 Specify a properties file.

3 Specify the location of the results file.

11 Using the Model Comparison Tool

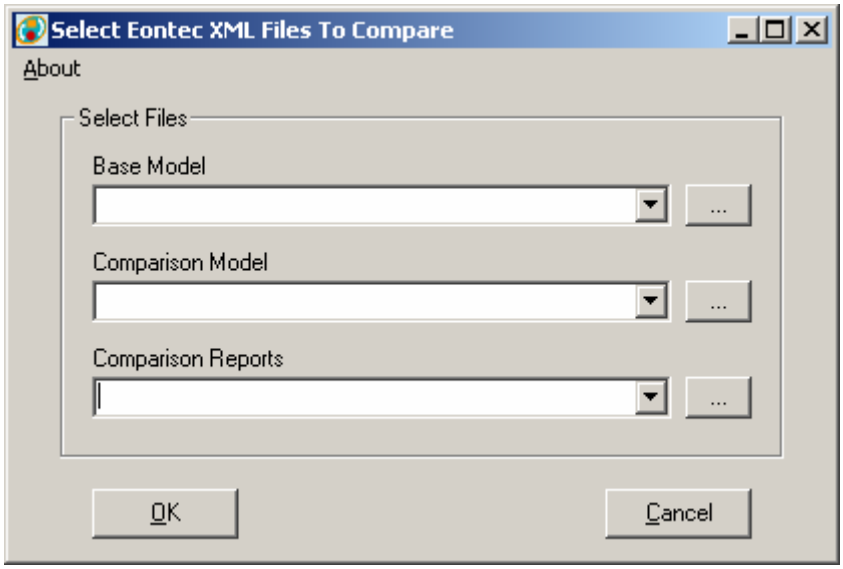
This chapter describes how to use the Model Comparison tool, which you use to compare two Siebel Retail Finance models. The chapter contains the following topics:

- Comparing Models
- Model Comparison Tool Error Messages
- Comparison Report Contents

Comparing Models

When you click the Model Comparison Tool icon on the Design Tools Palette, Figure 12 is displayed:

Figure 12. The Model Comparison Tool



This screen allows you to specify two XML files to be compared, and the location of the report files.

NOTE: The XML files must be valid model XML files. You must have exported the XML files, using the Model Exporter tool, from a Rational Rose Model that adheres to all design standards as specified in the Automated Methodology.

To compare two models

1. Click Model Comparison Tool on the Design Tools Palette.
2. Click the browse button beside the Base Model list.

3 Select the XML file for the base model, that is, the old version of the model that you want to compare against.

4 Click the browse button beside the Comparison Model list.

5 Select the XML file for the comparison model, that is, the newest version of the model.

6 Click the browse button beside the Comparison Reports list, and select the output location for the comparison report files.

7 Click OK to generate the Comparison Reports.

When the reports have been generated, a message is displayed indicating their output location. Depending on the size of the models being compared, the generation process can take a few minutes to complete.

Model Comparison Tool Error Messages

The errors messages described in the following sections can be displayed when using the Model Comparison Tool.

XML Parsing Error – An Invalid Character Was Found in Text Content

This message indicates that an invalid XML character was found in one of the selected XML files. Make sure that you run the Model Validator, and correct any invalid XML characters that are flagged in the Model Validator report.

The error message indicates the line number at which the invalid character was found. You can use a text editor to examine this line of the XML file to identify where the problem is located.

It is recommended that you update the actual Rational Rose Model to correct the problem rather than update the XML file. If you edit the XML file manually, the changes are not reflected in the Rational Rose Model and the problem occurs again the next time XML is exported from the model.

XML Parsing Error – The System Cannot Locate the Resource Specified

This message indicates that the EontecModel.dtd, which the tool uses to parse a Siebel model XML file was not found. The EontecModel.dtd file is installed in the same default location as the model XML file. However, an XML file generated on another machine that uses an older version of the Design Tools can have a reference to a different folder location in its DOCTYPE statement (located on the second line of the file).

To correct this problem, edit the XML file to change the location of the EontecModel.dtd to a valid location on your system.

CAUTION: Take great care when manually editing the XML files. If a tag is left open, or an invalid character is inserted by accident, the tool cannot read the file.

XML Parsing Error – The Base XML File Selected is Not of DOCTYPE EontecModel

This message indicates that an XML file was not of DOCTYPE EontecModel. It means that the file chosen was not a valid Siebel model XML file and was not exported from the Design Tools.

XML Parsing Error – A Duplicate Package Name Has Been Found in the Base XML

This message indicates that the XML file contains more than one object with the same package name. According to Automated Methodology standards, all classes must have unique package names. You must remove the duplicate package names from the design before you run the comparison tool again.

Comparison Report Contents

The following reports are produced by the Model Comparison Tool:

- Banking objects report (BankingObjects.html)
- Banking processes report (BankingProcesses.html)
- Common validations report (CommonValidations.html)
- Constants report (Constants.html)
- Parameter objects report (ParameterObjects.html)
- Validator lengths report (ValidatorLengths.html)

All reports can consist of an initial section, a summary section, and a detailed section. If there are no differences found for a particular report, the report contains only the initial section, followed by the message: `No Differences Found.`

Initial Section

All reports contain a header area containing the following information:

- **Date of Generation.** The date the model comparison report was produced.
- **Base Model.** The old version of the model.
- **Compared Model**. The latest version of the model.

The Summary Report Section

All reports can contain a summary section that contains summary totals for changes, additions, and deletions.

The Detailed Report Section

The following sections describe the contents of the detailed report section for each of the files.

Banking Object Report (BankingObjects.html)

The details section contains the following subsections:

- **Added Objects.** The name and package name of banking objects that have been added to the model.
- **Removed Objects.** The name and package name of banking objects that have been removed from the model.
- **Modified Objects.** Each modified object is represented as a table in the report. The object's name and package name are listed at the top of the table. The modifications to the object are listed in the following subsections:
  - **Modified Object Details.** Any properties of the banking object that have been modified.
  - **Modified Attributes.** Any attributes that have been modified. The values of the modified properties from both models are also listed (Value Before and Value After).
  - **Added Attributes.** Any attributes that have been added to the object.
  - **Removed Attributes.** Any attributes that have been removed from the object.
  - **Modified Methods.** Any methods that have been modified. Remember that it is possible for the same method name to exist with many signatures. Therefore, if you modify the signature of an existing method of a class using the Design Tools, the updated method is represented as a new method in the comparison report, while the previous version is represented as a removed method in the comparison report.
  - **Added Methods.** The name and signature of any methods that have been added to the object.
  - **Removed Methods.** The name and signature of any methods that have been removed from the object.

NOTE: The report does not list the properties, attributes, and methods of added objects and removed objects.

Banking Process Report (BankingProcesses.html)

The details section contains the following subsections:

- **Added Sessions.** The name and package name of sessions that have been added to the model.
- **Removed Sessions.** The name and package name of sessions that have been removed from the model.
- **Modified Sessions.** Each modified session is represented as a table in the report. The sessions name and package name are listed at the top of the table. The modifications to the session are listed in the following subsections:

- **Modified Object Details.** Any properties of the session that have been modified.
- **Modified Processes.** Any processes that have been modified. Remember that it is possible for the same process name to exist with many signatures. Therefore, if you modify the signature of an existing process on a session through the Design Tools, the updated method is represented as a new process in the comparison report, while the previous version is represented as a removed process in the comparison report.
- **Added Processes.** The name and signature of any processes that have been added to the object.
- **Removed Processes.** The name and signature of any processes that have been removed from the object.

NOTE: The report does not list the properties and processes of added sessions and removed sessions.

Common Validations Report (CommonValidations.html)

The details section contains the following subsections:

- **Added Objects.** The name and package name of CommonValidations objects that have been added to the model.
- **Removed Objects.** The name and package name of CommonValidations objects that have been removed from the model.
- **Modified Objects.** Each modified CommonValidations object is represented as a table in the report. The object's name and package name are listed at the top of the table. The modifications to the object are listed in the following subsections:
  - **Modified Validations**. Any validations that have been modified.
  - **Added Validations**. The name of any validations that have been added to the object.
  - **Removed Validations.** The name of any validations that have been removed from the object.

NOTE: The report does not list Validation methods of added and removed CommonValidations objects.

NOTE: The report only outlines changes to CommonValidations objects. Customized validations are stored on an attribute. Therefore, if an attribute's validation is customized, this appears in the banking objects report.

Constants Report (Constants.html)

The details section contains the following subsections:

- **Added Constants Classes.** The name and package name of any Constant classes that have been added to the model.
- **Removed Constants Classes.** The name and package name of any Constant class that has been removed from the model. There must only be one Constants class in the model (according to

Automated Methodology standards). If the package name of this class is changed, the report indicates that the class has been removed and a new one added.

- **Added Constants.** Any constants that have been added to the Constants class.
- **Removed Constants.** Any constants that have been removed from the Constants class.
- **Modified Constants.** Each modified constant is represented as a table in the report. The constant's name is listed at the top of the table. The modifications to the constant are listed in the following subsections:
  - **Modified Constant Details.** Information about any changes to the data type, which is the only detail that can change.
  - **Added Values.** The name of any constant values that have been added to the constant.
  - **Removed Values.** The name of any constant values that have been removed from the constant.

NOTE: Do not confuse the term Constant with the term Constants Class. A Constant represents a value or list of values. An example of a Constant is ACCOUNT_TYPE. The ACCOUNT_TYPE constant could contain Constant Values of Current Account and Deposit Account. The Constants Class is the Class that stores these constants in the model.

Parameter Objects Report (ParameterObjects.html)

The details section contains the following subsections:

- **Added Objects.** The name and package name of parameter objects that have been added to the model.
- **Removed Objects.** The name and package name of parameter objects that have been removed from the model.
- **Modified Objects.** Each modified object is represented as a table in the report. The object's name and package name are listed at the top of the table. The modifications to the object are listed in the following subsections:
  - **Modified Object Details.** Any properties of the parameter object that have been modified.
  - **Modified Attributes.** Any attributes that have been modified. The values of the modified properties from both models are also listed (Value Before and Value After).
  - **Added Attributes.** Any attributes that have been added to the object.
  - **Removed Attributes.** Any attributes that have been removed from the object.

NOTE: The report does not list the Validation methods of added and removed objects.

Validator Lengths Report (ValidatorLengths.html)

The details section contains the following subsections:

- **Added Validator Lengths.** Any validator lengths that have been added to the model.
- **Removed Validator Lengths.** Any validator lengths that have been removed from the model.

- **Modified Validator Lengths.** The name and details of each modified validator length.

NOTE: The only validator length detail that can change is the length.

NOTE: Validator lengths are stored on the Constants class in the Siebel XML representation of the design model. If the package name of the Constants class changes, it is considered to have been removed and a new one added. If this has happened, the Model Comparison Tool does not examine the validator lengths for differences and only generates the initial section of the report, followed by the message: `The Constants Class has been removed or renamed.`