Design Tools Guide

Version 2004.5

September 2004

Siebel Systems, Inc., 2207 Bridgepointe Parkway, San Mateo, CA 94404

Contents

1 What's New in this Release

The following changes have been introduced in the Siebel Retail Finance Design Tools, Version 2004.5:

Topic	Description
The Class Builder now supports a wide range of Data Types, page 14	Attributes can now be defined with a wide range of data types including Java Primitives, Parameter Objects and Classes. The existing set of data types such as String and Boolean are still supported.
The Model Validator validations are now more configurable/customizable, page 38	Users can now decide which validations to check for when using the Model Validator. Each validation can be switched on or off by means of a properties file. Additionally, users have the ability to specify their own custom Rose script.

2 Design Tools Overview

The Siebel Retail Finance Design Tools comprise of the following:

2.1 Class Builder

The Class Builder was developed to aid in the design of Siebel standard entity EJBs. It makes use of already existing attributes within the project to assist in Class definition. The Class Builder also automatically updates background properties within the Rose model. These help in the automatic entity code generation.

2.2 Method Definer

The Method Definer was developed to aid the design of Siebel standard entity and session EJBs. It updates the Siebel properties of the methods including the overview and behaviour. The Method Definer has made it easier to create new methods, as it is easier to add parameters to the signature of a method and define the return types

2.3 Validation Definer

The Validation Definer was developed to aid in the design of Siebel standard entity EJBs. The Validation Builder creates Common Validators that are held in the .mdl file. These Common Validators are then available to all designers when creating the attributes of entities. This means that the designer can select an existing Validator, which speeds up the process. The Validators are held in a single class, which gets generated by the code generator

2.4 Session Builder

The Session Builder was developed to aid in the design of standard session EJBs. The Session Builder also automatically updates background properties within the Rose model. These help in the automatic entity code generation.

2.5 Attribute Class Definer

The Attribute Class Definer ensures that Attributes defined for a particular class are made available for use throughout the model. By running the Attribute Class Definer, the Attributes package is updated with the details of all Attributes defined in the model.

2.6 Design Documentation Builder

The Design Documentation Builder was developed to produce standard design documents. This is made up of Project directories that contain Financial Objects, Sessions, Parameter Objects, Parameter Object Factories, External Interfaces, and Java Class documents.

2.7 Model Exporter

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

2.8 Model Validator

The Model Validator was developed so that designers could ensure that their designs meet design standards. This is to ensure 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.

2.9 Model Comparison Tool

The Model Comparison Tool compares the differences between two Siebel Models. The tool was designed to aid designers and developers as follows: to facilitate handover of updated Rose models to the development team, to help measure progress during the design phase, to produce input to the review of design artifacts and to help synchronization of streams within the design process. It produces 6 reports in HTML format - one for each of the following: financial components, financial processes, paramater objects, common validations, validator lengths and constants. These comparison reports can be used to identify changes to a model during the design, development and testing phases of a project.

3 Class Builder

3.1 Introduction

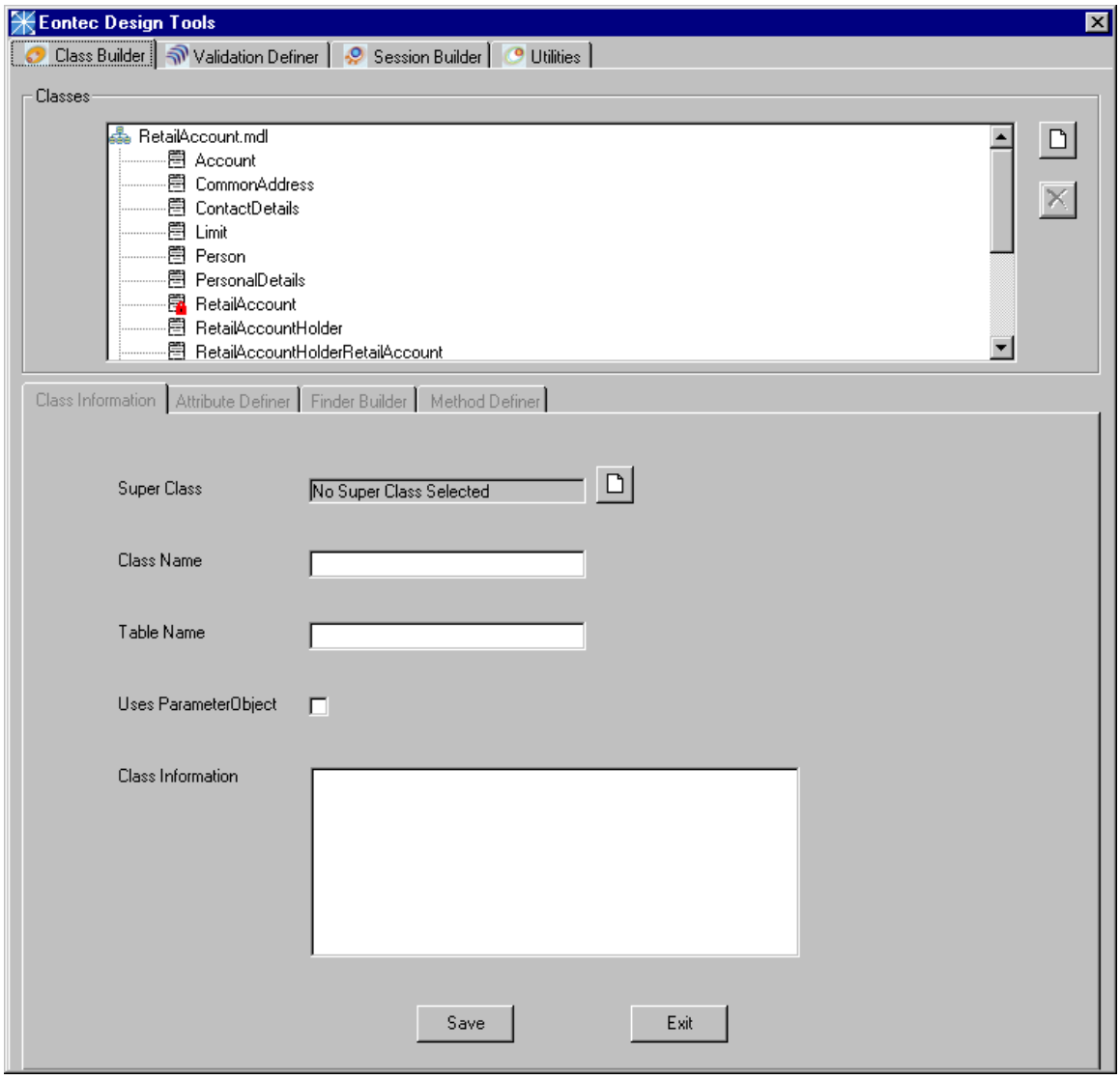
The Class Builder was developed to aid in the design of standard entity EJBs. It makes use of already existing attributes within the project to assist in Class definition. The Class Builder also automatically updates background properties within the Rose model. These help in the automatic entity code generation.

3.2 Using the Class Builder

Once a new model has been created using the Framework then the entities may be designed. This tool is launched from the Tools>Siebel menu in Rose. The Class Builder appears on the Design Tools Palette.

The Class Builder enables the user to create new Class Packages in the Banking Objects section of the model. It provides the ability to create/delete domain layer packages and classes.

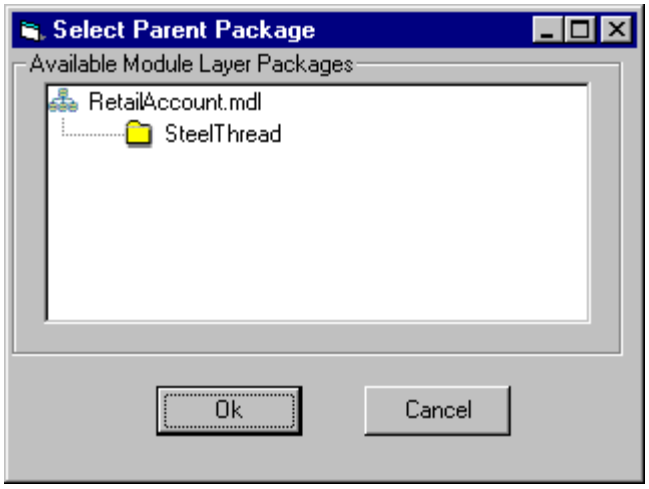
On loading, all of the existing classes will be shown in the Class List. There is only one class per package so the creation and maintenance of the packages will be done in the background. The elements in the Class information section become active when a user selects an existing class or chooses to create a new class. Any classes that are contained within a write-protected package will appear with a lock symbol next to them. This is shown for the RetailAccount class in the diagram below.



3.3 Class Information Tab

3.3.1 Create New Class

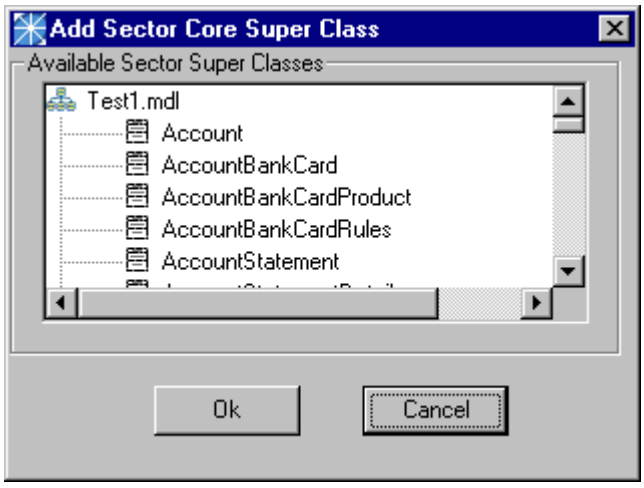
To create a new class the  button should be selected. When this is selected the following screen is launched:



This enables the user to define the BO Grouping that they wish to add the Entity to. The Class Information section becomes active at this point. The user may enter the Class name (this is a mandatory field), table name, and class overview into the relevant text fields.

The user may also select whether the class uses a non-functional parameter object. This will create a background property that indicates to the code generator that a Parameter Object containing all of the attributes of the class should be created. This will be used in `create` and `amend` methods of the object. If this is not checked on creating the class it can be amended afterwards and saved with the list of non-functional parameter objects being refreshed with the class name.

The architecture is a Four Layer object architecture. The possible super classes of the new class are all contained in the 'Available Super Class' list as shown below:



In the Production release this list contains all of the Sector Core Layer Classes. In the Delivery release this contains all of the Module Layer Classes. To save the information the User should select 'Save'. The information will also be saved if the User selects one of the other tabs e.g. FinderBuilder. In both cases a new Class is added to the selected BO Grouping of the Rose model. 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 there are a number of methods added to the classes. These are `create`, `amend` and `delete` on the module layer class, and `findByPrimaryKey` on the Domain Layer. In the Delivery release the new Class is only added at the Domain Layer and all of the Default methods are added to this class.

3.3.2 Amend existing Class

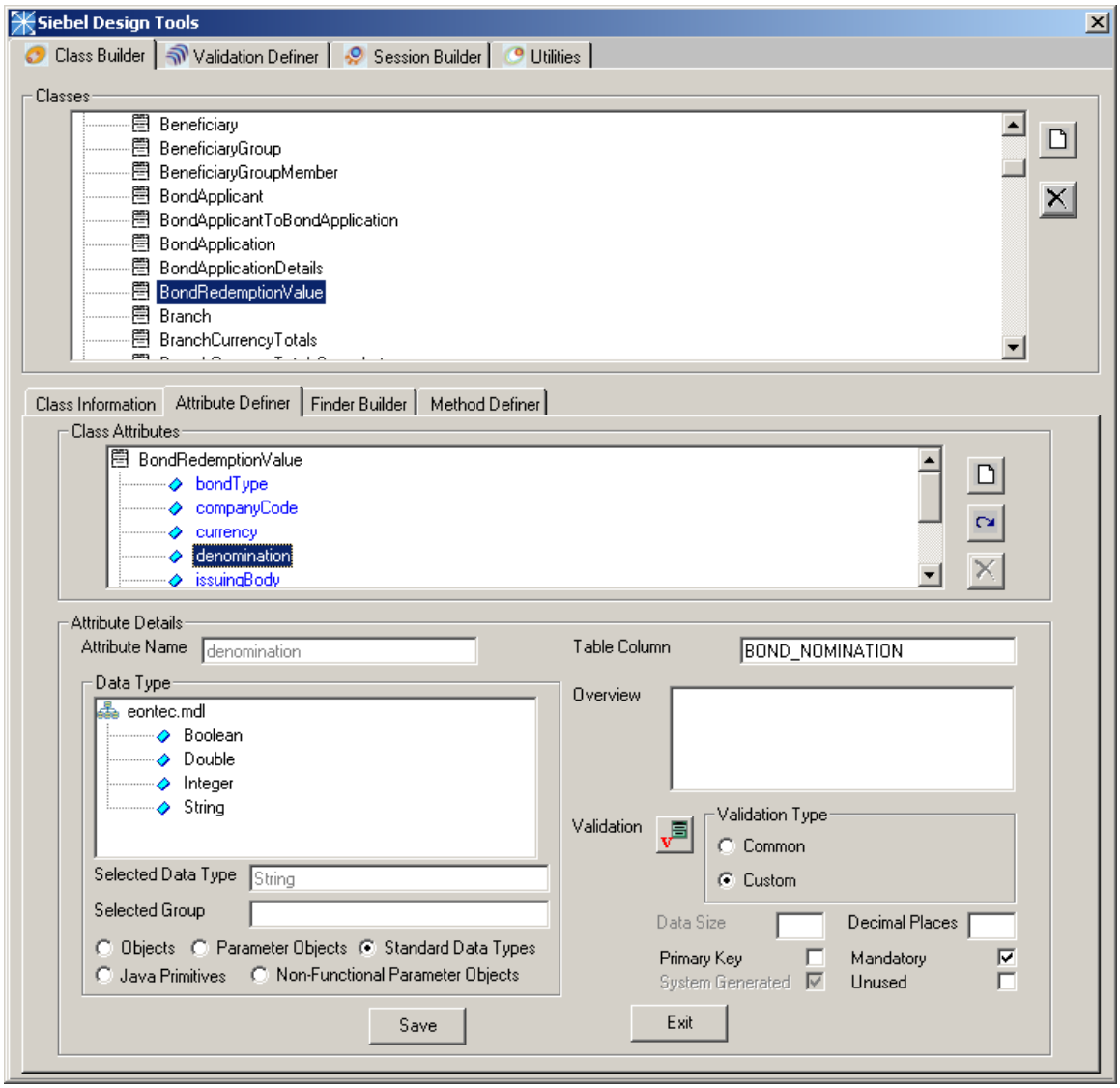
When a Class is selected from the list the Class Information section becomes active. The user may amend the fields (except for the Name field) in this section. In order to amend the Class the user should select 'Save'. This will update the Class with the newly entered information.

3.3.3 Delete existing Class

When a Class is selected from the list the  button becomes enabled. If a user selects this, the selected class is deleted from the model. In the Production release both the Module and Domain Layer Classes are deleted. However in the Delivery release only the Domain Layer Class is released.

3.4 Attribute Definer Tab

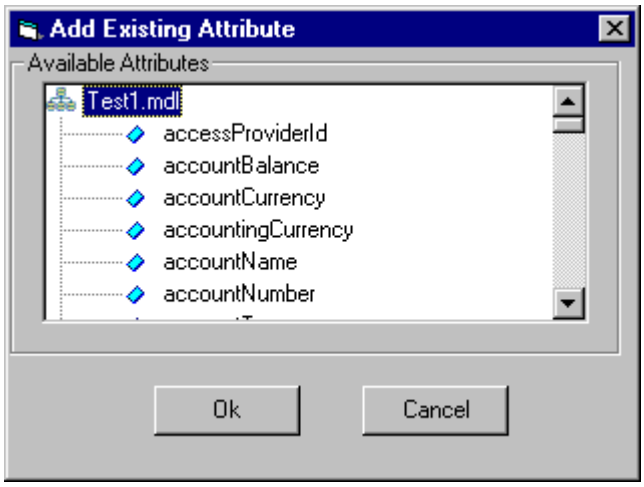
The attributes of the selected class may also be defined by selecting the 'Attribute Definer' tab. When this is selected the class builder screen looks as below:



In the Production release the classes that are selected are in the Module Layer. In this case the Class Attributes List only shows attributes of the Module Layer and its parent classes. In the Delivery release the classes that are selected are in the Domain Layer. In this case the Class Attributes List shows attributes of the Domain Layer and its parent classes. In both cases the attributes that are in the actual class are shown in black. The attributes of the parent classes are shown in blue. If an attribute is from a super class then the following fields are not amendable: Attribute Name, Overview and Data Type. All other parts of the Attribute Details section are amendable. For an attribute in the actual class all fields are amendable.

3.4.1 Create New Attribute

To create a new attribute either  or  should be selected. If  is selected then the following screen is launched:

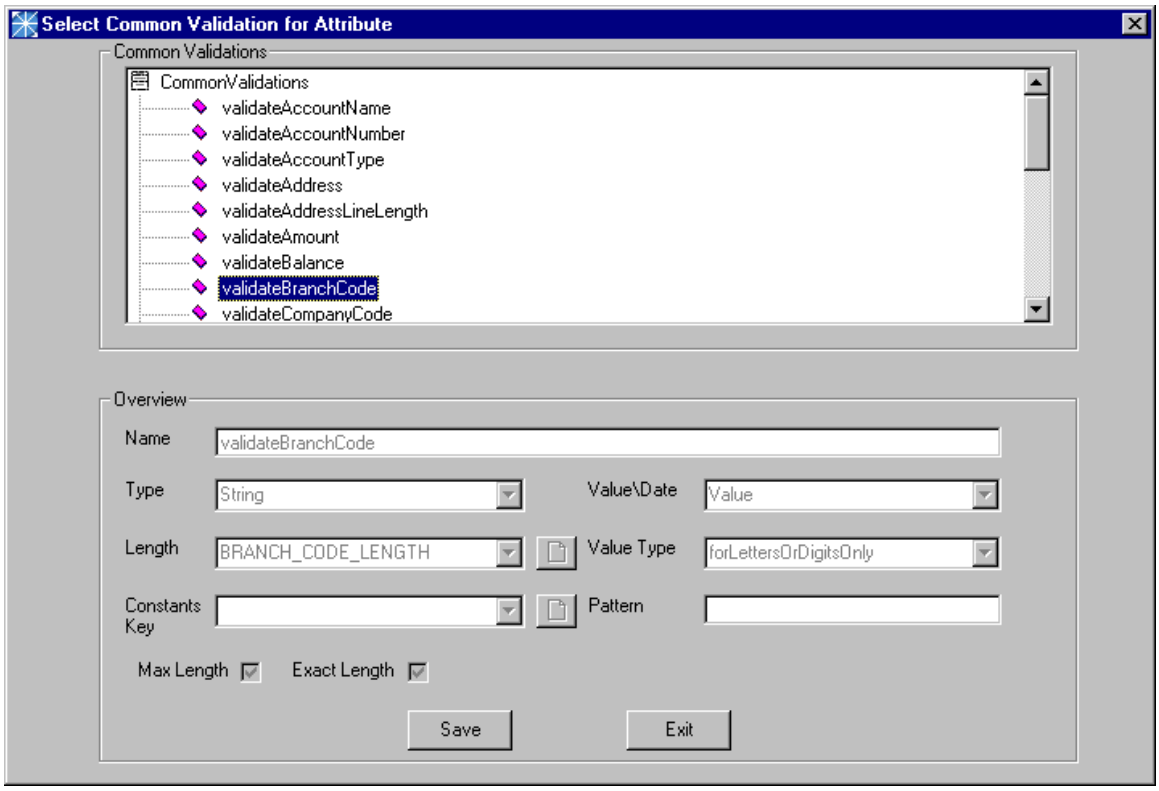


This enables the user to add an existing application or system attribute to the entity. The advantage of this is that the attribute definition is complete. This makes the design quicker and ensures consistency throughout the model.

If  is selected the Attribute Details section becomes active. The user may enter the 'Attribute Name', 'Table Column' and 'Overview' into the relevant text fields. The data type of the attribute may be selected by clicking on a data type in the tree view. The view may be changed by clicking on the radio buttons in the data type section: Objects, Parameter Objects, Standard Data Types, Java Primitives, Non-Functional Parameter Objects

If one of the Standard Data Types (Boolean, Double, Integer, String) is selected, the User will have the option to assign validation to the attribute. However, for all other data types, validation will be disabled.
It should be noted here that the tool does not allow attributes of the same name to be defined on an entity. If the attribute is defined with a data type of double then the 'Decimal Places' field is enabled. The user may enter the number of decimal places in the text box. The attribute will default to have Optional validation or can be given Mandatory validation by selecting the 'Mandatory' check box. The attribute can be defined as a Primary Key by selecting the 'Primary Key' check box. This will default the attribute as mandatory and will enable the 'System Generated' check box. The User may select a primary key field to be system generated if the attribute is taken from the system on the create of an entity. In the case of Standard Data Types, the user may also define the Validator for the attribute at this stage. To do this they must select whether they want to use a 'Common' or 'Custom' validator then select  .

If Common is selected the following screen is launched:

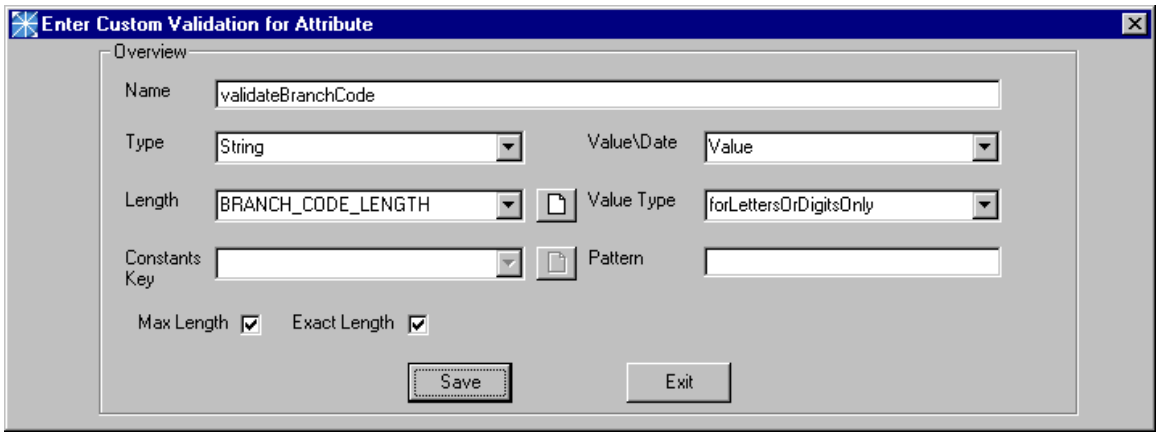


This allows the user to select from the list of existing Common Validations. When a validator has been selected the details of it are shown in the 'Overview' section. If the user should select 'Save' and the datatype of the validator is not valid a message will instruct the user to select a validator with the correct datatype to match the attribute datatype. Below are the valid datatype matches for a Validator:

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

Once an appropriate validator has been chosen the user should select 'Save'. This will return the user to the Attribute Definer. To exit the screen the user should select 'Exit'.

If Custom is selected the following screen is launched:



On entry the 'Name' field will default to 'validateAttributeName', the 'Type field will be defaulted to 'String' the 'Value\Date' field will default to 'Value' and the 'Value Type' field will default to 'forLettersOrDigitsOnly'. The user may configure the validator as follows:

First the Validator Type should be selected. This can be either Boolean, Number or String and can be chosen from the drop down. The other fields in the 'Overview' section will be disabled depending on the Type chosen:

- If 'Boolean' is selected the all other fields become disabled.
- If "Number" is selected then the 'Length', 'Max Length' and 'Exact Length' fields are enabled.
- If 'String' is selected then the 'Value\Date', Value Type' and 'Length' fields are enabled.

When 'Date Or Time' is selected in the 'Value\Date' drop down then the user may select 'Supply Pattern' or 'System Date Pattern' from the 'Value Type' drop down. If 'Supply Pattern' is selected then the 'Pattern' field becomes enabled and the user may enter a specific date or time pattern e.g. yyyy:mm:dd. If 'System Date Pattern' is selected, then the 'Pattern' field defaults to 'System'. In both cases the 'Length', 'Max Length' and 'Exact Length' fields are disabled.

When 'Value' is selected in the 'Value\Date' drop down then the user may select 'Constant', 'forDigitsOnly', 'forLettersOnly', 'forLettersOrDigitsOnly' or 'forLettersOrDigitsOrWhiteSpacesOnly' from the 'Value Type' drop down. Where the Validator is a constant and 'Constant' is chosen then the 'Length', 'Max Length' and 'Exact Length' fields are disabled and the 'Constants Key' field is enabled. In this case the user may select an existing Constants Key from the drop down or may choose to add a new constant to the system by selecting the  button along side the drop down. In the other three cases the User may select a validator length from the 'Length' field or create a new length using  . The User may also select 'Max Length' or 'Exact Length' if the validator so requires.

Once the attribute has been completely defined 'Save' must be selected. It is only then that all of the information is stored within the model.

3.4.2 Amend

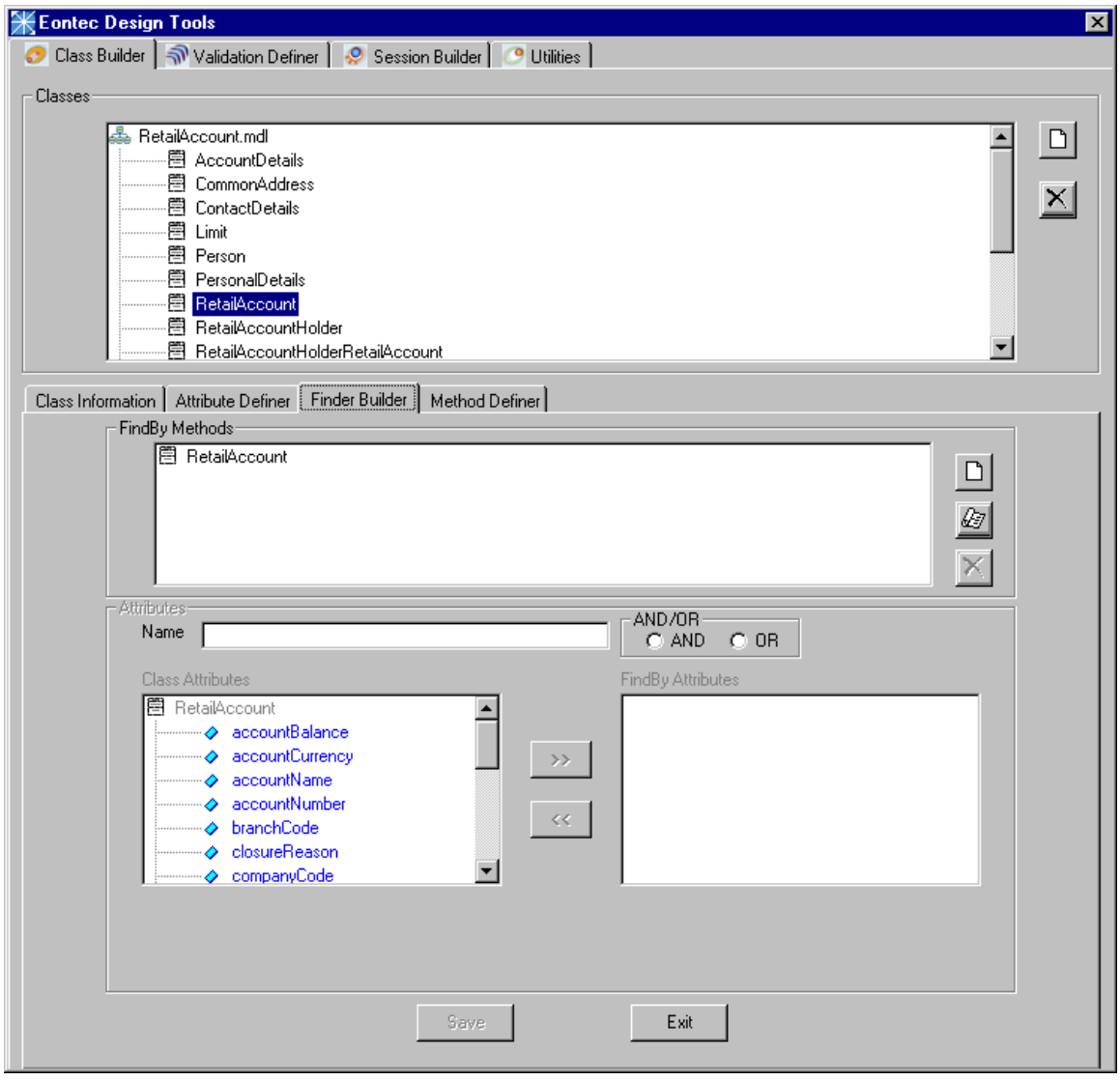
To amend an existing attribute the user should select the attribute from the list and update the 'Attribute Details' section as described above. If an attribute is from a super class then the following fields are not amendable: Attribute Name, Overview and Column Format. All other parts of the Attribute Details section are amendable. For an attribute in the actual class all fields are amendable.

3.4.3 Delete

When a class attribute is selected from the list the  button becomes enabled (it will be disabled for super class attributes). If a user selects this, the selected attribute is removed from the entity.

3.5 Finder Builder Tab

The finder methods of the selected class may also be defined by selecting the 'Finder Builder' tab. When this is selected the class builder screen looks as below:



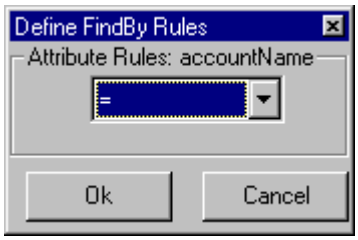
The Finder Builder allows the user to create, amend and delete `findBy` methods. It also allows the user to create, amend and delete complex finder methods. In the Production release all of the Finder methods in the list will be from the Module and will be amendable. In the Delivery release the Module Layer Finder methods will appear in blue and will not be amendable. The Implementation Finder methods will be shown in black and will be amendable.

3.5.1.1 Create simple FindBy

To add a new finder method to the selected entity the user should select  . This enables the Attributes section of the Finder Builder Tab. To create a simple `findBy` method the user must select whether the method is an 'AND' or an 'OR' finder. Selecting the radio buttons in the Attributes section of the screen does this. An 'OR' finder will find objects that fulfill any of the parameters passed in. An 'AND' finder will find any objects that fulfill the combination of parameters. Once the user has selected the type of finder, an attribute from the list containing all of the Class attributes should be chosen and '>>' should be selected. This will update the 'FindBy Attributes' list with the selected attribute. To remove an attribute from the `findBy` method, the attribute should be selected from the 'FindBy Attributes' list and '<<' should be selected. Once the `findBy` has been properly defined 'Save' should be selected.

3.5.1.2 Create complex FindBy

To add a new complex finder method to the selected entity the user should select  . This enables the Attributes section of the Finder Builder Tab. To create a complex `findBy` method the user must first enter a name for the finder. Then the user should select whether the method is an 'AND' or 'OR' finder. Selecting the radio buttons in the Attributes section of the screen does this. An 'OR' finder will find objects that fulfill any of the parameters passed in. An 'AND' finder will find any objects that fulfill the combination of parameters. Once the user has selected the type of finder, an attribute from the list containing all of the Class attributes should be chosen and '>>' should be selected. This will update the 'FindBy Attributes' list with the selected attribute. To remove an attribute from the `findBy` method, the attribute should be selected from the 'FindBy Attributes' list and '<<' should be selected. To amend the rules on the parameters the user should double click on one of the attributes in the 'FindBy Attributes' list. This launches the screen shown below:



The user may select the rules to be equal to, less than, less than or equal to, greater than, greater than or equal to, not equal to or range. Once the user is happy that the correct rule has been applied to the attribute 'Ok' should be selected. Once the `findBy` has been properly defined 'Save' should be selected.

3.5.1.3 Amend

To amend an existing finder method, the user should select the method from the ‘FindBy Methods’ list. For all finder method the user may add or remove attributes for the findBy. The user may also change the finder method from an ‘AND’ method to an ‘OR’ method or vice versa. In the case of complex finder methods, the user may also amend the rules on each parameter.

3.5.1.4 Delete

To remove a `findBy` method from a Class the method should be selected from the ‘FindBy Methods’ list and  should be selected.

3.5.2 Method Definer

When ‘Method Definer’ is selected on the ‘Class Builder’ (first) screen the Method Definer is launched. This enables the User to define methods on the class. A method is defined by giving it a name, a signature and a return type. The Method Definer also allows the User to define a more detailed behaviour of a method. The Method Definer is detailed in the Method Definer document.

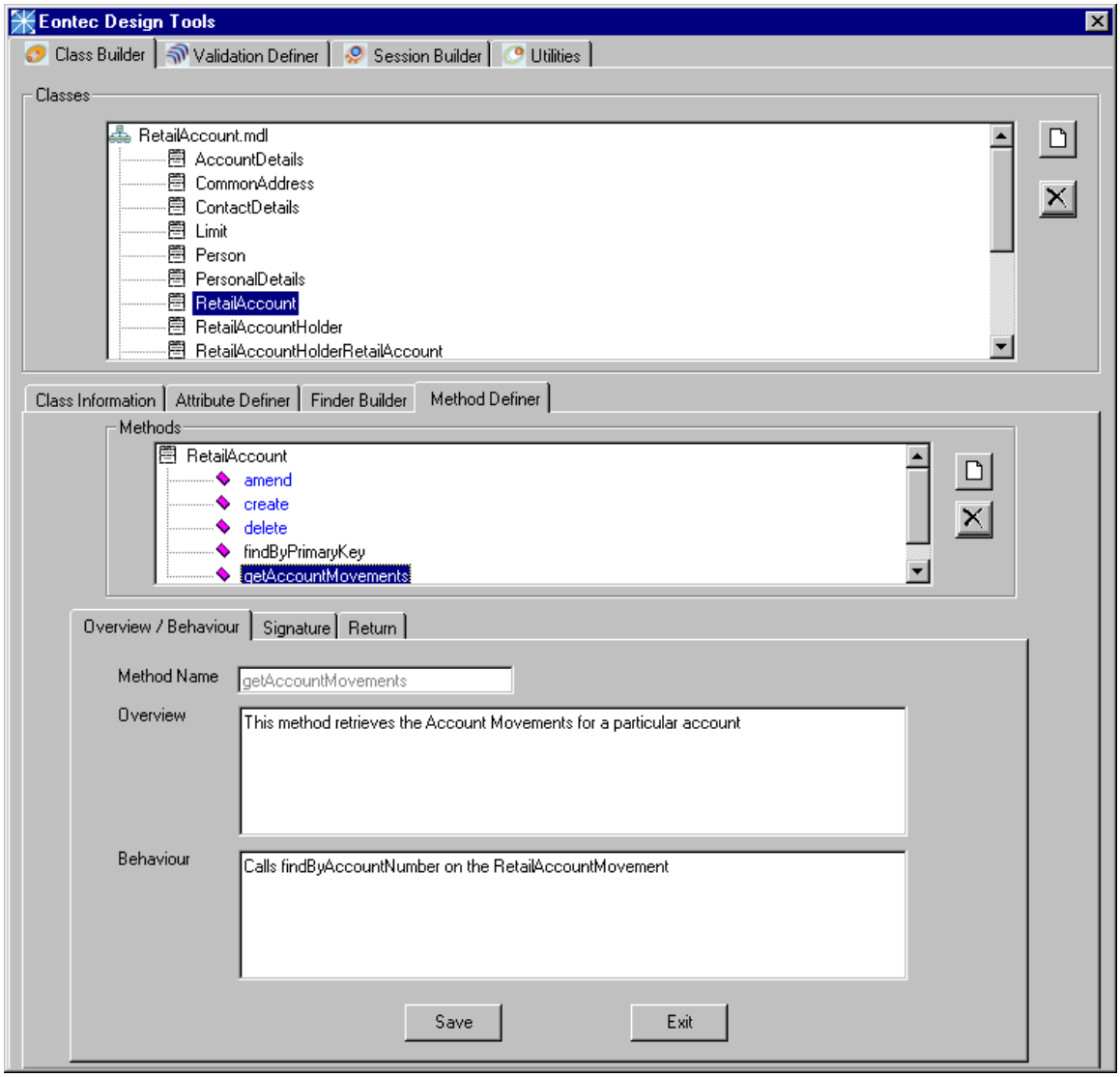
4 Method Definer

4.1 Introduction

The Method Definer was developed to aid the design of standard entity and session EJBs. It updates the properties of the methods including the overview and behaviour. The Method Definer has made it easier to create new methods, as it is easier to add parameters to the signature of a method and define the return type.

4.2 Using the Method Definer

The Method Definer can be used from the Class Builder or the Session Builder. In both cases the functionality that is offered is the same. When using the Class Builder the Method Definer tab becomes active when a Class is selected. If the tab is chosen the screen appears as shown below:



4.3 Overview/Behaviour tab

4.3.1 Create New Method

To create a new method the  icon should be selected on the 'Method Definer' tab. The Overview/Behaviour section becomes active at this point. The user enters the Method name (this is a mandatory field), the method overview and an overview of the method behaviour into the relevant text fields. The 'Methods' list is updated with the new method.

4.3.2 Amend Method

Methods of the actual class will be shown in black and methods of the super classes will be shown in blue. Only class methods may be amended. To amend a method it should be selected from the list. This will enable the user to amend the detail tabs. Once these have been updated the User should select 'Save'. Note that the text "* Deprecated" appears beside the method name of any selected method that is deprecated.

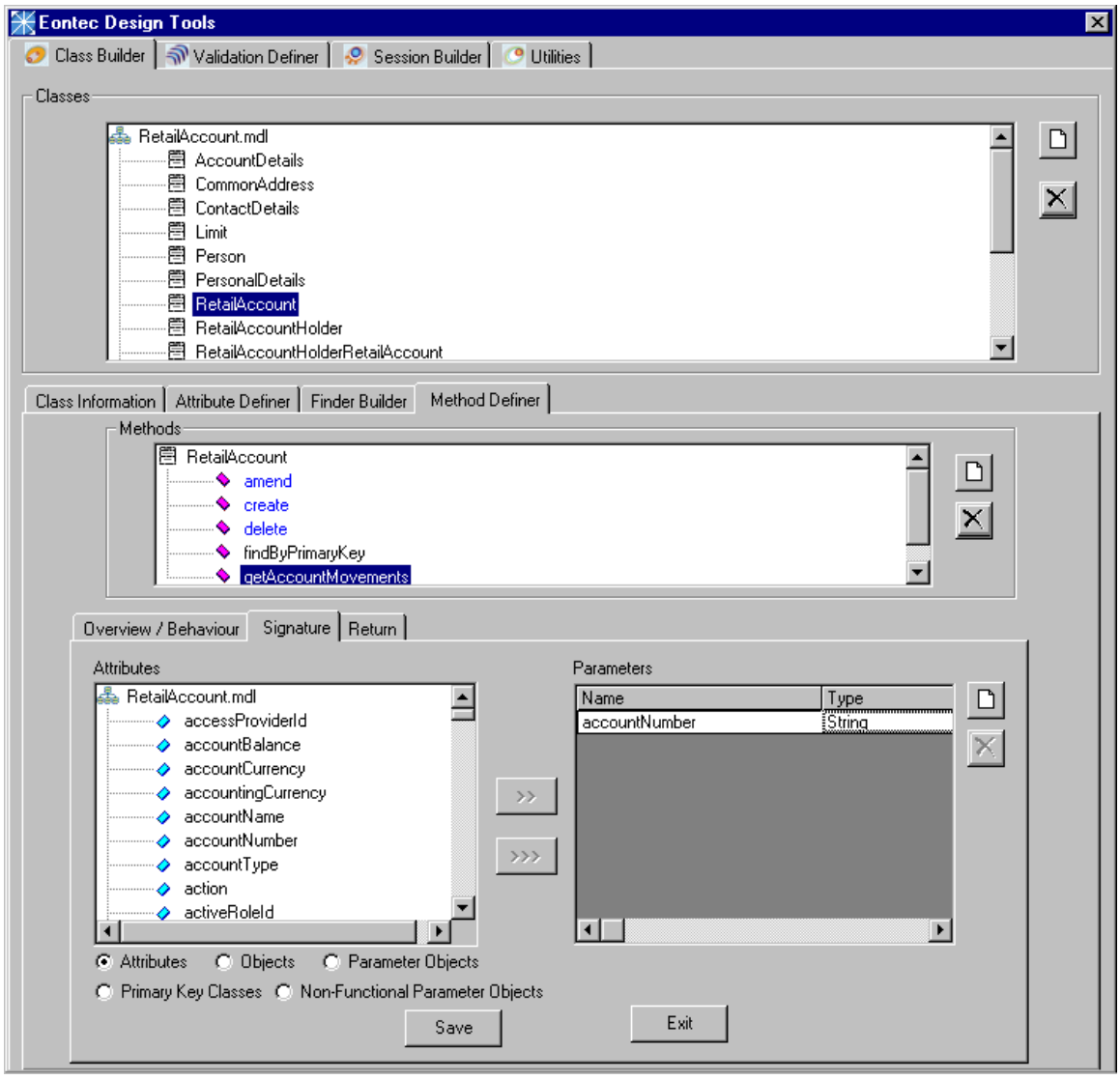
4.3.3 Delete Method

When a Method is selected from the list the  button becomes enabled. If a user selects this, the selected method is deleted from the Class.

4.4 Signature Tab

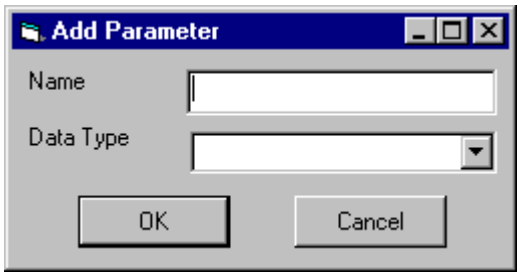
4.4.1 Update Signature

The parameters of the selected method may be updated by selecting the 'Signature' tab. If this is selected the screen appears as follows:



4.4.1.1 Adding a Parameter

The Method Signature Definer allows the User to select parameters for the method that has been selected. There are a number of lists comprising of all of the attributes, Objects, Parameter Objects, Primary Key Classes and Non-Functional Parameter Objects that are in the project. To add a parameter, choose from these lists and select '>>'. In the case of Objects, Parameter Objects, Primary Key Classes and Non-Functional Parameter Objects '>>>' may be selected. This will add a collection containing these objects to the parameter list. When adding a Parameter Object or Non-Functional Parameter Object an 'Impl' suffix is added to the name of the parameter object. A parameter may also be added by selecting the  icon. If this is selected then the following screen is launched:



This allows the input of 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. Once all of the parameters have been added to the signature 'Save' must be selected at the bottom of the tab.

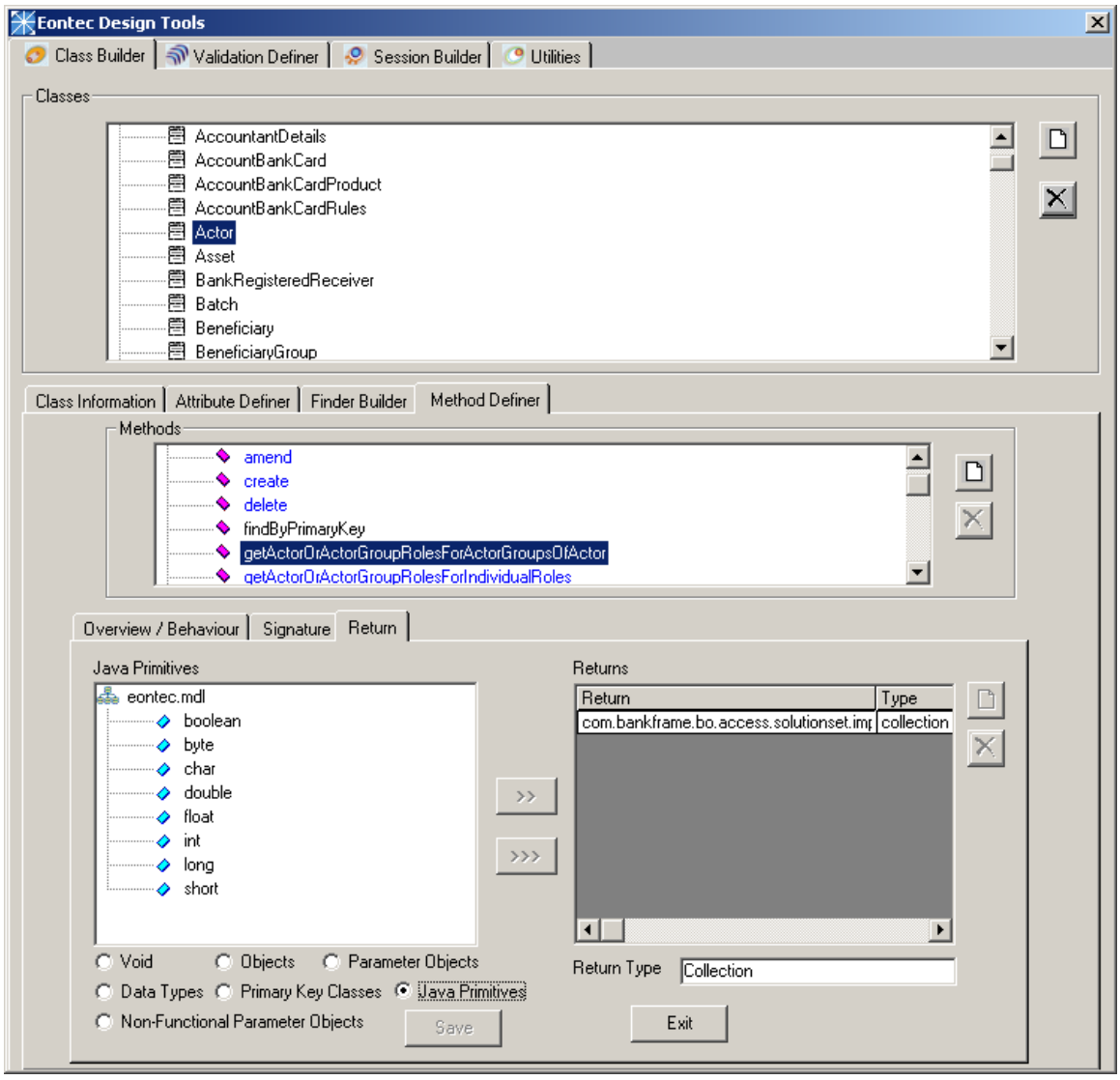
4.4.1.2 Removing a parameter

To remove a parameter from the parameter list it should be selected from the 'Parameters' list. Then select the  icon and the parameter will be deleted.

4.5 Return Tab

4.5.1 Update Return

The return of the selected method may be updated by selecting the 'Return tab. If this is selected the screen appears as below:



4.5.1.1 Adding a return

The Method Return Definer allows the User to select primitives and objects as returns for the method that has been selected. There are a number of lists comprising of all of the Objects, Parameter Objects and Primary Key Classes that are in the project. The User may also define the method as a data type or `Void` by selecting the relevant radio buttons. To add a data type or object, choose from the relevant lists and select either '>>' to add a single selected item or '>>>' to add a collection. This will add a collection containing these objects to the return. In this case the return will be a collection such as a vector or enumeration. Once the return has been completely added to the method 'Save' must be selected at the bottom of the tab.

4.5.1.2 Removing a return

To remove a primitive or object from the return list it should be selected from the 'Returns' list. Then select the  icon and the selected item will be deleted.

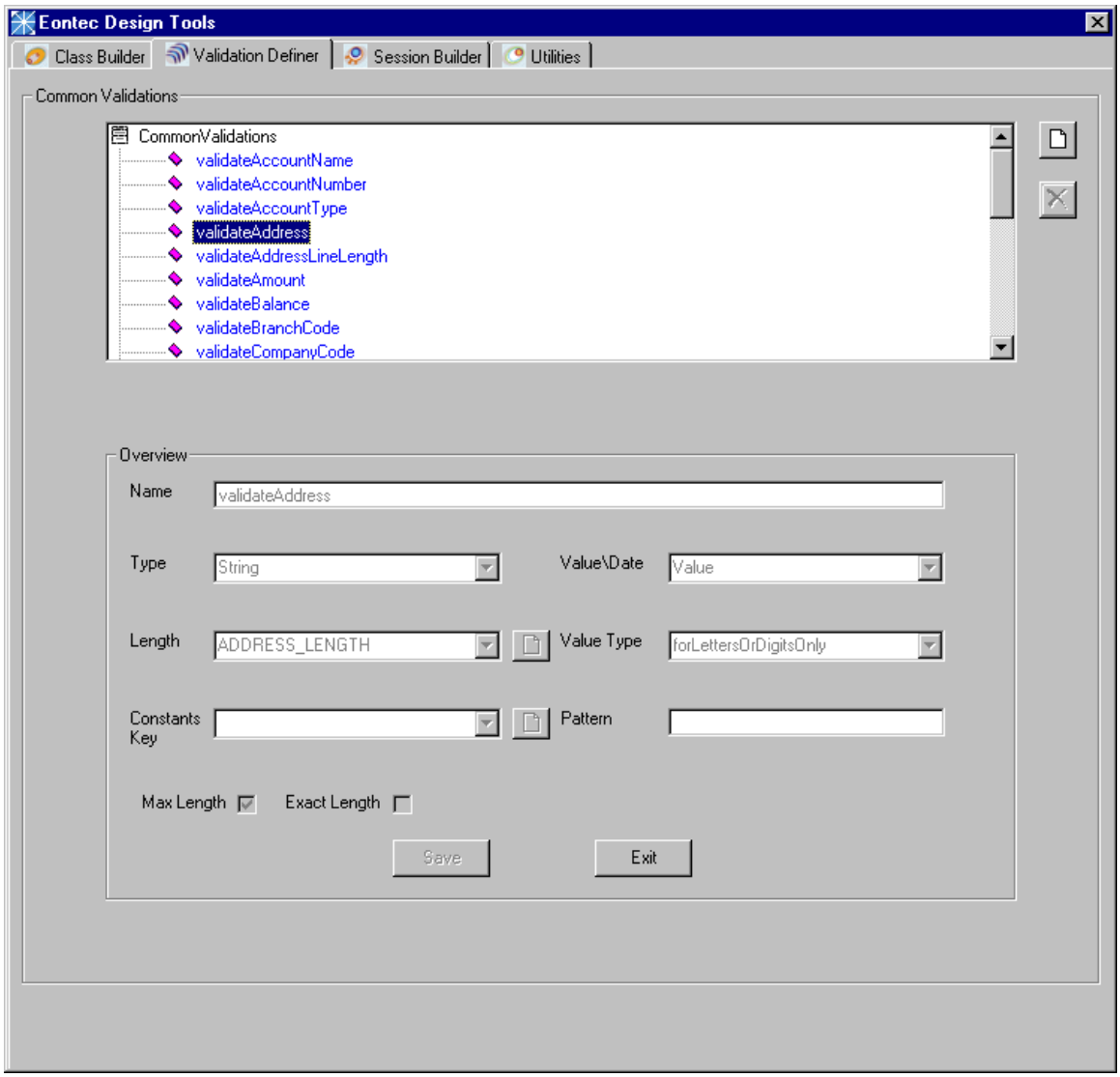
5 Validation Definer

5.1 Introduction

The Validation Definer was developed to aid in the design of standard entity EJBs. The Validation Builder creates Common Validators that are held in the `.mdl` file. These Common Validators are then available to all designers when creating the attributes of entities. This means that the designer can select an existing Validator, which speeds up the process. The Validators are held in a single class, which gets generated by the code generator

5.2 Using the Validation Definer

This tool is launched from the Tools>Siebel>Design Tools menu in Rose.



The Validation Definer enables the user to create new Common Validators in the model. It also provides the ability to delete and amend selected existing Validators. To view the properties of an existing Validator is should be selected from the list. The details will appear in the 'Overview' section. In the Production release all of the Common Validators in the list will be from the Solutionset Common Validator and will be amendable. In the Delivery release the Solutionset Common Validators will appear in blue and will not be amendable. The Implementation layer Common Validators will be shown in black and will be amendable.

5.2.1 Create Validation

To create a new Common Validator select the  icon. The 'Overview' section becomes enabled. The new Validator name should be entered into the text field as follows: "validateName" e.g. "validateBranchCode". Next the Validator Type should be selected. This can be either Boolean, Number or String and can be chosen from the drop down list. The other fields in the 'Overview' section will be disabled depending on the Type chosen.

- If 'Boolean' is selected then all other fields become disabled.
- If "Number" is selected then the 'Length', 'Max Length' and 'Exact Length' fields are enabled.
- If 'String' is selected then the 'Value\Date', Value Type' and 'Length' fields are enabled.

When 'Date Or Time' is selected in the 'Value\Date' drop down list then the user may select 'Supply Pattern' or 'System Date Pattern' from the 'Value Type' drop down. If 'Supply Pattern' is selected then the 'Pattern' field becomes enabled and the user may enter a specific date or time pattern e.g. yyyy:mm:dd. If 'System Date Pattern' is selected, then the 'Pattern' field defaults to 'System'. In both cases the 'Length', 'Max Length' and 'Exact Length' fields are disabled.

When 'Value' is selected in the 'Value\Date' drop down list then the user may select 'Constant', 'forDigitsOnly', 'forLettersOnly', 'forLettersOrDigitsOnly' or 'forLettersOrDigitsOrWhiteSpacesOnly' from the 'Value Type' drop down list. Where the Validator is a constant and 'Constant' is chosen then the 'Length', 'Max Length' and 'Exact Length' fields are disabled and the 'Constants Key' field is enabled. In this case the user may select an existing Constants Key from the drop down list or may choose to add a new constant to the system by selecting the  button along side the drop down list. Note that the constant type must match the validation type, which is String in this case. In the other three cases the user may select a validator length from the 'Length' field or create a new length using the  icon. The user may also select 'Max Length' or 'Exact Length' if the validator so requires.

5.2.2 Amend

To amend an existing common validator the user should select the validator from the list and update the 'Overview' section as described above. The 'Name' field in this instance is disabled so that the common validator names cannot be amended. This is done so that references to existing common validators are not broken

5.2.3 Delete Existing

When ‘Delete Existing’ is selected the selected Common Validator is removed form the Model. Care must be taken in deleting existing validators as entity attributes may already reference these validators.

6 Session Builder

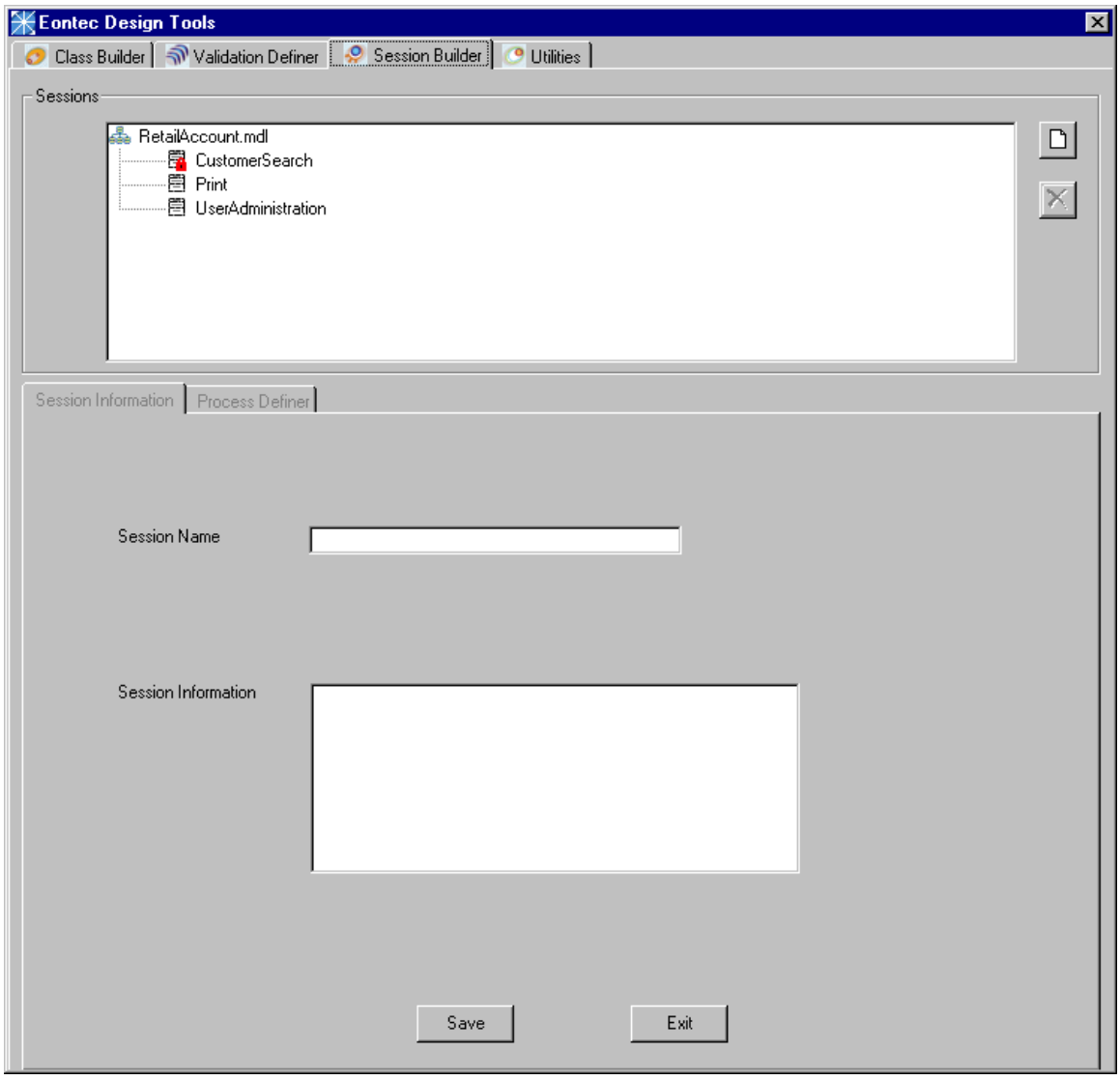
6.1 Introduction

The Session Builder was developed to aid in the design of standard session EJBs. The Session Builder also automatically updates background properties within the Rose model. These help in the automatic entity code generation.

6.2 Using the Session Builder

Once a new model has been created using the Framework then the entities may be designed. This tool is launched from the Tools>Siebel menu in Rose. The Session Builder appears on the Design Tools Palette. The Session Builder enables the user to create new Class Packages in the Banking Objects section of the model. It provides the ability to create/delete domain layer packages and classes.

On loading, all of the existing sessions will be shown in the Session List. There is only one class per package so the creation and maintenance of the packages will be done in the background. The elements in the Session information section become active when a user selects an existing class or chooses to create a new class. Any classes that are contained within a write-protected package will appear with a lock symbol next to them. This is shown for the CustomerSearch class in the diagram below.



6.3 Session Information Tab

6.3.1 Create New Session

To create a new session the  button should be selected. When this is selected the following screen is launched:

This enables the user to define the BP Grouping that they wish to add the Session to. The Session Information section becomes active at this point. The user may enter the Session name (this is a mandatory field) and overview into the relevant text fields. The solution works within a Two Layer session architecture. To save the information the User should select 'Save'. The information will also be saved if the User selects one of the other tabs e.g. Process Builder. In both cases a new Class is added to the selected BP Grouping of the Rose model. 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.

6.3.2 Amend existing Session

When a Session is selected from the list the Session Information section becomes active. The user may amend the fields (except for the Name field) in this section. In order to amend the Session the user should select 'Save'. This will update the Session with the newly entered information.

6.3.3 Delete existing Session

When a Session is selected from the list the  button becomes enabled. If a user selects this, the selected session is deleted from the model. In the Production release both the Module and Domain Layer Sessions are deleted. However in the Delivery release only the Domain Layer Session is released.

6.3.4 Process Definer

When 'Process Definer' is selected on the 'Session Builder' (first) screen the 'Method Definer' is launched. This enables the User to define processes on the session. A process is defined by giving it a name, a signature and a return type. The Method Definer also allows the User to define a more detailed behaviour of a process. The Method Definer is detailed in the Method Definer document.

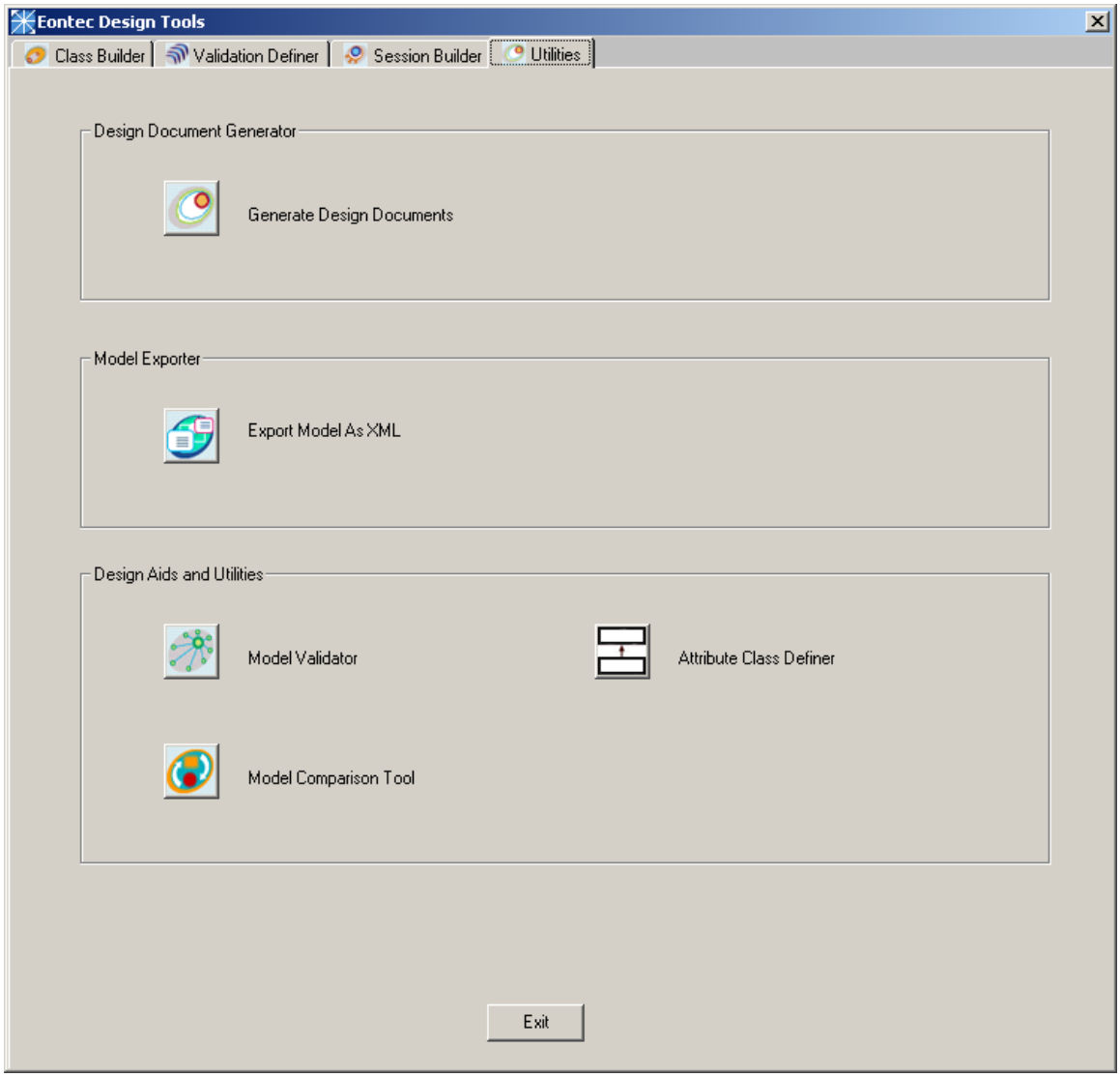
7 Design Aids & Utilities

7.1 Introduction

The Design Aids & Utilities were developed to 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.

7.2 Using the Design Aids & Utilities

All of the Design Aids & Utilities can be launched from the Tools>Siebel menu within Rose. Each is discussed below.



7.2.1 Generate Design Documents

This utility is described in the Design Documentation Builder document.

7.2.2 Export Model as XML

This utility is described in the Model Exporter document.

7.2.3 Model Validator

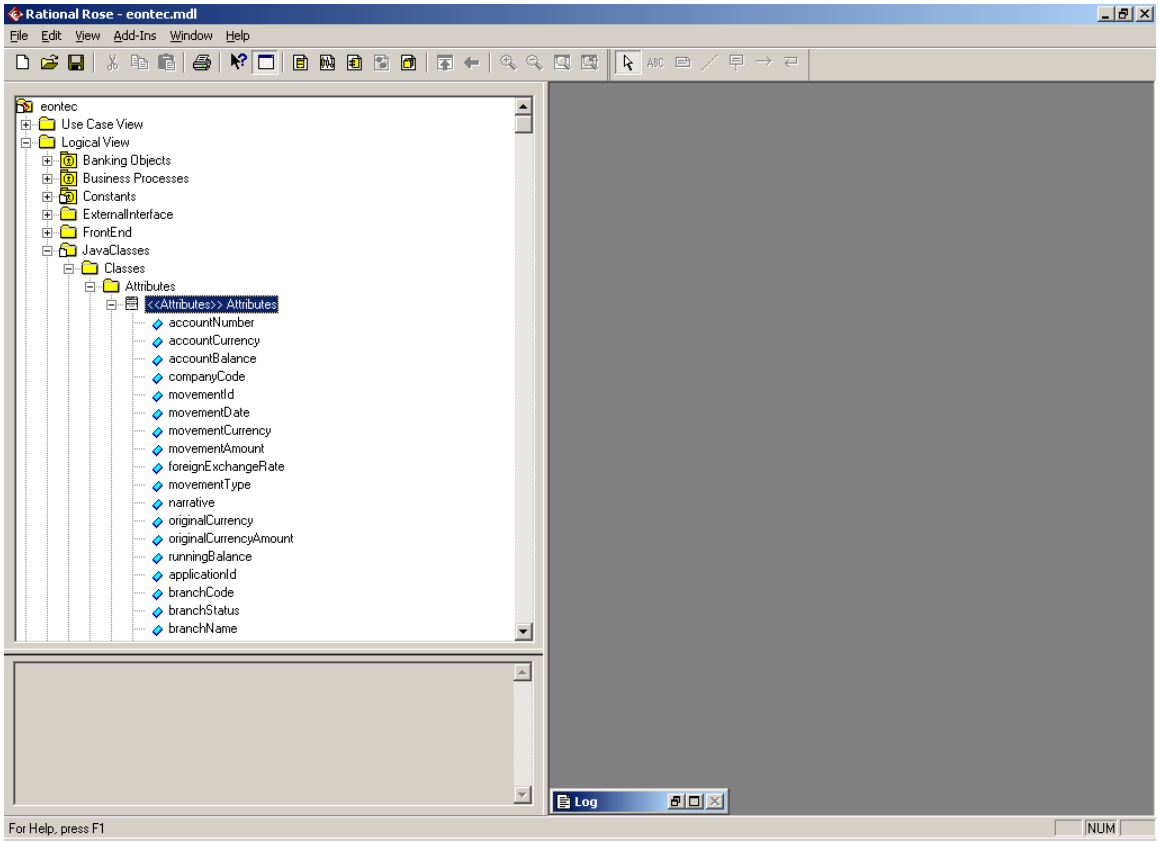
This utility is described in the Model Validator document.

7.2.4 Model Comparison Tool

This utility is described in the Model Comparison Tool document.

7.2.5 Attribute Class Definer

The Attribute Class Definer ensures that Attributes defined for a particular class are made available for use throughout the model. To run the Attribute Class Definer select its icon on the Utilities tab. By running the Attribute Class Definer, the Attributes package is updated with the details of all Attributes defined in the model.



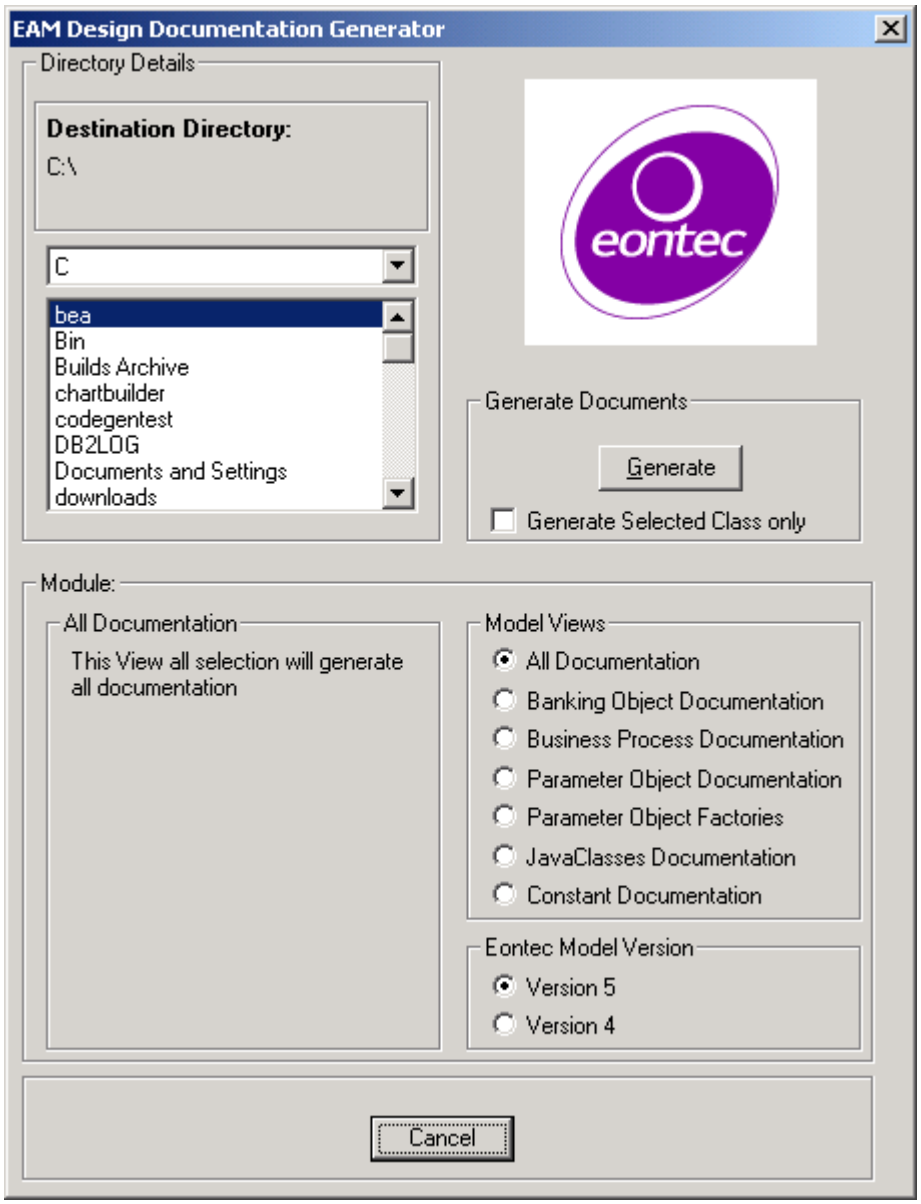
8 Design Documentation Builder

8.1 Introduction

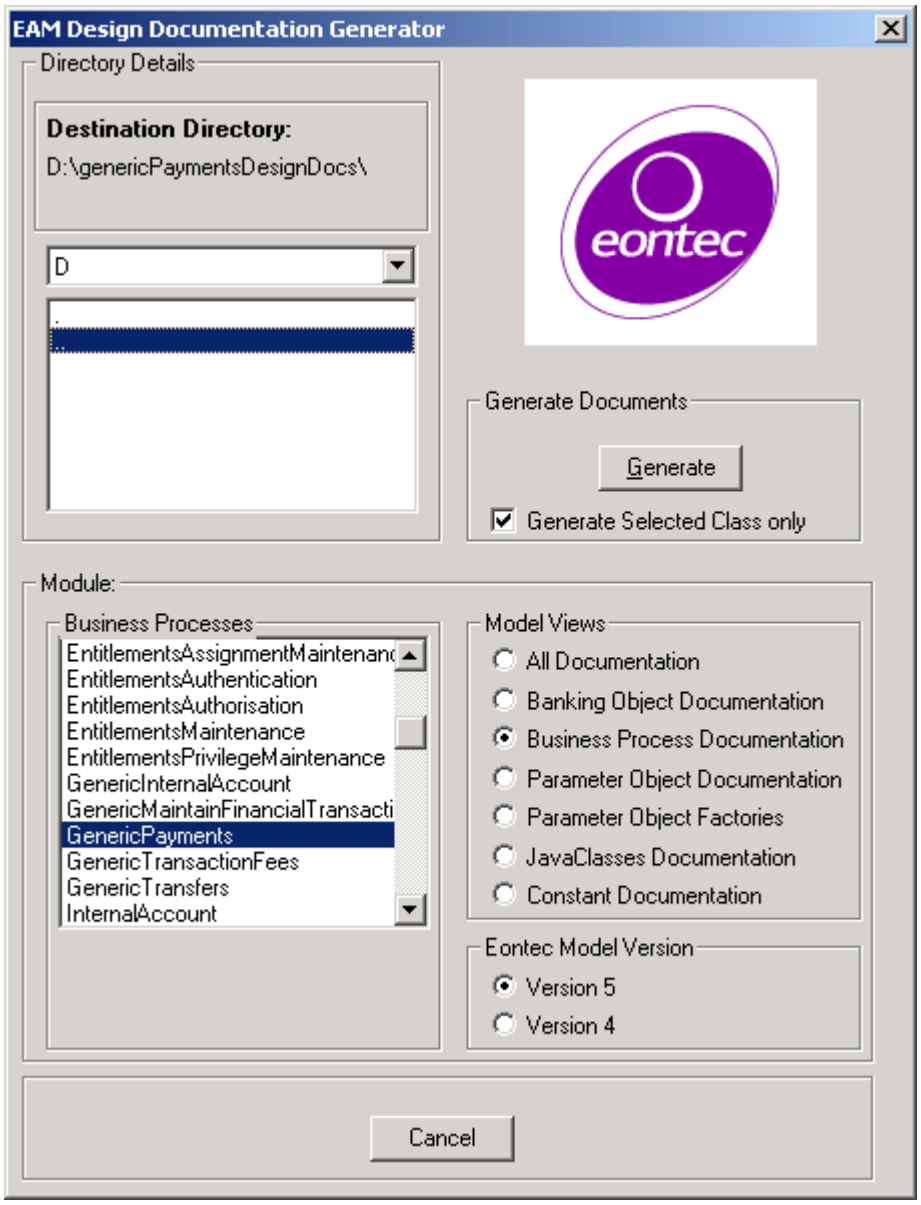
The Design Documentation Builder was developed to produce standard design documents. The generator produces a directory structure that is compliant to these standards. This is made up of Project directories that contain Financial Objects, Sessions, Parameter Objects, Parameter Object Factories, External Interfaces, and Java Class documents.

8.2 Using the Design Documentation Builder

Once the designer has completed the design phase for a requirement the Design Documentation Builder may be used to create standard design documents. The Document Generator is launched from the Tools>Siebel menu within Rose.



Once the Generator has launched, the user must select the Destination Directory for the documents (using the Drop-Down box and List box in the top left corner). The generator also provides the ability to select which documents should be generated. Using the 'Model Views' radio buttons on the right the user may select to generate all of the design documents, or only a selection of design documents, which may be Banking Object documents, Business Process documents, Parameter Object documents, Parameter Object Factory documents, JavaClasses documents or Constant documents. Alternatively the user may select to generate the documents of a single class by selecting a class from the list of classes displayed when one of the 'Model Views' options is selected and checking the 'Generate Selected Class only' box.



The user must also select the Version of the model that is being worked on (either Version 4 or Version 5). Clicking the 'Generate' button will generate the design documents and output same to the destination directory.

*An important note regarding Design document generation is that there is a Windows restriction of 256 characters in the fully qualified file name of any file, i.e. the file name with the full folder and sub folders names all added up together. Overloading operation 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 will exit at this point. In order to get over this issue, try generating the documents in a folder off the c:\ or d:\ root drive, or shorten the class and/or operation names.

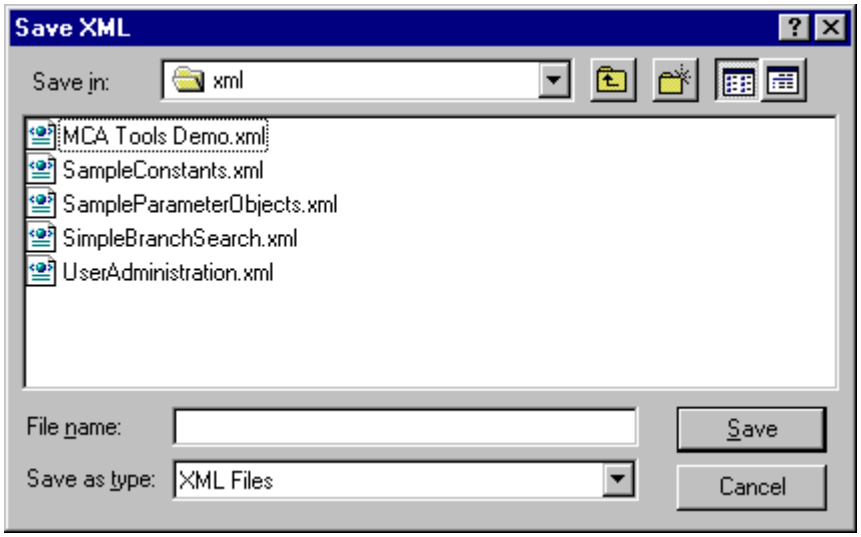
9 Model Exporter

9.1 Introduction

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

9.2 Using the Model Exporter

Once a model has been completed, or a module has been taken from the repository then the XML may be exported. This tool is launched from the Tools>Siebel menu in Rational Rose.



When the Model Exporter is launched the above screen appears. The User can enter the name of the XML file they want to generate or select a current file to overwrite. Once this is done the Save button can be clicked and the file will be generated to the chosen location.

9.3 Changing DTD location

The XML generated by the Model Exporter references a DTD file. The DTD can be referenced in the XML as a file or a URL. It is important that the DTD location is accessible when the XML file is passed to other Retail Finance tools. For example, if the DTD location referenced in the XML is a network drive or a URL and the user has no network connectivity, other Retail Finance tools may not be able to read the file. If the user wishes to change the DTD referenced they can do so by choosing Tools>Siebel menu (Change DTD Location) in Rational Rose.

This is a prompt for the User to select where the DTD they want to use for the export is located. If 'New File' is selected then the User may select a DTD from a directory. The User may also choose a URL for the DTD. The URL entered for the DTD must end in `/EontecModel.dtd`.

10 Model Validator

10.1 Introduction

The Model Validator was developed so that designers could ensure that their designs meet design standards. This is to ensure that the code can be generated quickly and easily from the models when the design phase is over. There are a number of checks which the Model Validator does, outlined below.

10.2 Using the Model Validator

The Model Validator tool is launched from the Tools->Siebel menu in Rose.

The user can edit the validations.properties file to determine which validations are selected. To remove a validation, the symbol # may be inserted at the beginning of the line. The validations.properties file has the following format:

```
# Custom Rose Script
customModelValidations.ebx
# Attribute Validations
AttributeMandatoryValidation
AttributeDataSize
# AttributeDataSizeNumeric
AttributeValidatorMethodDefined
```

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

When the Model Validator is invoked, the properties file is read. If a validation is “commented out” it will not be used to validate the model.
The tool performs the following checks on the model and produces a HTML report:

Section	Validation Description	Validation Code
General Errors		
	Check that the documentation field in the BankingObject Catagory is not blank	GeneralBankingObjectCategoryDocumentation
	Check that the BankingObject Category contains a Domain Layer	GeneralBankingObjectCategoryDomain
	Check that the BankingObject Category contains a Module Layer	GeneralBankingObjectCategoryModule
	The documentation field in the Domain layer packages (stereotype DomainPackageObject)	GeneralBankingObjectDomainLayerCategoriesDocumentatio

	must contain part of the Java namespace	n
	The documentation field in the Module layer packages (stereotype ModulePackageObject) must contain part of the Java namespace	GeneralBankingObjectModuleLayerCategoriesDocumentation
	BusinessProcess Category documentation must contain com.bankframe.bp	GeneralBusinessProcessCategoryDocumentation
	BusinessProcess Category must contain a Domain Layer	GeneralBusinessProcessCategoryDomain
	BusinessProcess Category must contain a Module Layer	GeneralBusinessProcessCategoryModule
	Check that the Domain Layer Categories contain qualified name documentation	GeneralBusinessProcessDomainLayerCategoriesDocumentation
	Check that the Module Layer Categories contain qualified name documentation	GeneralBusinessProcessModuleLayerCategoriesDocumentation
Validator Errors		
	Check that the model contains a Validator Category	ValidatorCatagoryExists
	Validator Category documentation must contain com.bankframe.validator	ValidatorCatagoryDocumentation
	Validator Category must contain a Domain Layer	ValidatorCatagoryDomainExists
	The Domain Layer Validator Category must contain qualified name documentation	ValidatorCatagoryDomainDocumentation
	The Domain Layer Validator Category must contain at least one class	ValidatorDomainClasses
	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	ValidatorModuleStereotype

	the SolutionsetValidator package	
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 Contants class must have a stereotype of “Constants”	ConstantsClassStereotype
Attributes 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		
	Check that no Object starts with a small letter	classstartslowercaseCheck
	Check that no Object is misspelled or contains illegal characters	classillegalcharacterCheck
	Check that no Object inherits from multiple classes	classMultipleInheritanceCheck
	Check that no Object has Duplicate associations	classDuplicateAssocationCheck
	Check that no Solution Set Objects exist without corresponding Domain Package Objects	classdomainlayerclassCheck
	Check that no Session contains BPDs without a process on the session of the same name	
	Check that every package name matches its session name	classdifferentpackagenameCheck
	Check that findByPrimaryKey is correctly spelt	classFindByPKCheck
	Check that the Stereotype is correct in all cases	classSterotypeCheck
	Check that every entity has at least one primary key attribute defined for it	classPrimaryKeyCheck
	Check that every entity has at least one Attribute	classNoAttributesCheck
	Check that every session has at least one Method	

Attribute Errors		
	Check that attribute data size is defined	attributeDataSizeCheck
	Check that attribute table column name is defined	attributreColumnCheck
	Check that attribute data type is defined	attributeDataTypeCheck
	Check that attribute validation has been defined (stereotype)	attributeMandatoryValidationCheck
	Check that no attribute validation has been defined (non-stereotype)	attributeValidatorCheck
	Check that no attribute starts with a capital letter	attributeLowercaseCheck
	Check that no attribute is misspelled or contains illegal characters	attributeIllegalCharacterCheck
	Check that no Object has duplicate attributes	classDuplicateCheck
	Check that attribute data size is defined with a numeric value	attributeDataNumericCheck
	Ensure that the table column of the attribute is not defined in the wrong part of the model. Note: The value for theAttribute.GetPropertyValue("ETHOS","eontablecolumn") should be blank – the table column information is no longer stored in this field.	attributeOldColumnCheck
	Ensure that there is no overwriting in the Solution Set	
Function Errors		
	Check that every function overview has been completed	functionOverviewCheck
	Check that every function behavior has been completed	functionBehaviourCheck
	Check that every function's parameters have been properly defined	functionParameterCheck
	Check that no function starts with a capital letter	functionLowercaseCheck
	Check that no function is misspelled or contains illegal characters	functionNameIllegalCharacterCheck
	Check that no function behaviour contains illegal characters	functionBehaviourIllegalCharacterCheck
	Check that no function overview contains illegal characters	functionOverviewIllegalCharacterCheck
	Make sure all the parameters for a findBy are attributes of the object the findBy is on	functionFindByCheck

	Check that the function has a return type specified	functionReturnTypeCheck
	Check that any method that overwrites an existing method has the same return type as the original method	functionReturnOverwriteCheck
Session Errors Module Layer		
	Check that every Session name does not contain an illegal character	SessionModuleIllegalCharacter
	Check that the Session name starts with a lower case letter	SessionModuleStartsLowerCase
	Check that the Session has a corresponding class on the domain layer	SessionModuleDomainLayer
	Check that the Session Package Name does not differ from the Class name	SessionModuleDifferentPackageName
	Check that the Session has the correct stereotype	SessionModuleStereotype
	Check that the Session contains at least one operation	SessionModuleOperationsExist
	Check that no method is overwriting a method with a different return type	SessionModuleMethodOverwritingDifferentReturn
BPD Errors – Module Layer		
	Check that no BPD contains module layer objects	BPDModuleNonDomainLayerObject
	Check that no BPD contains undefined classes	BPDModuleUndefinedObject
	Check that the BPD has a process associated with it on the Session	BPDModuleMethodAssociated
	Check that the BPD does not have an undefined message	BPDModuleUndefinedMessage
Process Errors Module Layer		
	Check that the Process is not named with an illegal character	ProcessModuleNameIllegalCharacter
	Check that every Process overview has been completed	ProcessModuleOverviewDefined
	Check that every Process overview does not contain illegal characters	ProcessModuleOverviewIllegalCharacter
	Check that every Process behavior has been completed	ProcessModuleBehaviourDefined
	Check that every Process behaviour does not	ProcessModuleBehaviourIlleg

	contain illegal characters	alCharacter
	Check that every Process response has been completed	ProcessModuleReturnDefined
	Check that no function starts with a capital	ProcessModuleStartsLowerCase
	Check that all method arguments have a type	ProcessModuleParameterValidType
Session Errors Domain Layer		
	Check that every Session name does not contain an illegal character	SessionDomainIllegalCharacter
	Check that the Session name starts with a lower case letter	SessionDomainStartsLowerCase
	Check that the Session is not inherited from multiple objects	SessionDomainInheritedMultiple
	Check that the Session Package Name does not differ from the Class name	SessionDomainDifferentPackageName
	Check that the Session has the correct stereotype	SessionDomainStereotype
	Check that the Session contains at least one operation	SessionDomainOperationsExist
	Check that no method is overwriting a method with a different return type	SessionDomainMethodOverwritingDifferentReturn
BPD Errors – Domain Layer		
	Check that no BPD contains module layer objects	BPDDomainNonDomainLayerObject
	Check that no BPD contains undefined classes	BPDDomainUndefinedObject
	Check that the BPD has a process associated with it on the Session	BPDDomainMethodAssociated
	Check that the BPD does not have an undefined message	BPDDomainUndefinedMessage
Process Errors Domain Layer		
	Check that the Process is not named with illegal character	ProcessDomainNameIllegalCharacter
	Check that every Process overview has been completed	ProcessDomainOverviewDefined
	Check that every Process overview does not contain illegal characters	ProcessDomainOverviewIllegalCharacter

	Check that every Process behavior has been completed	ProcessDomainBehaviourDefined
	Check that every Process behaviour does not contain illegal characters	ProcessDomainBehaviourIllegalCharacter
	Check that every Process response has been completed	ProcessDomainReturnDefined
	Check that no function starts with a capital	ProcessDomainStartsLowerCase
	Check that all method arguments have a type	ProcessDomainParameterValidType

10.3 Selection of Model Validator Properties File

The user has the facility to select a properties file, according to the project they are working on. When the user clicks the Model Validator button on the utilities tab, the user is prompted to select a properties file.

This allows the user to predefine multiple properties files, according to the requirements of different projects.

The model validator file selection and its directory will be saved locally as the default properties file.

10.4 Running the Model Validator with the Default Properties File

The user has the option to run the model validator with the default properties file. This is implemented by means of a new button – 'Model Validator – Default'

The last properties file that was used to validate the model will automatically be selected. If the functionality is being run for the first time and the default properties file does not exist in its saved directory, the user will be prompted to select the properties file manually, as described in the previous section.

11 Model Comparison Tool

11.1 Introduction

The Model Comparison Tool compares the differences between two models. The tool was designed to aid designers and developers in the following ways:

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

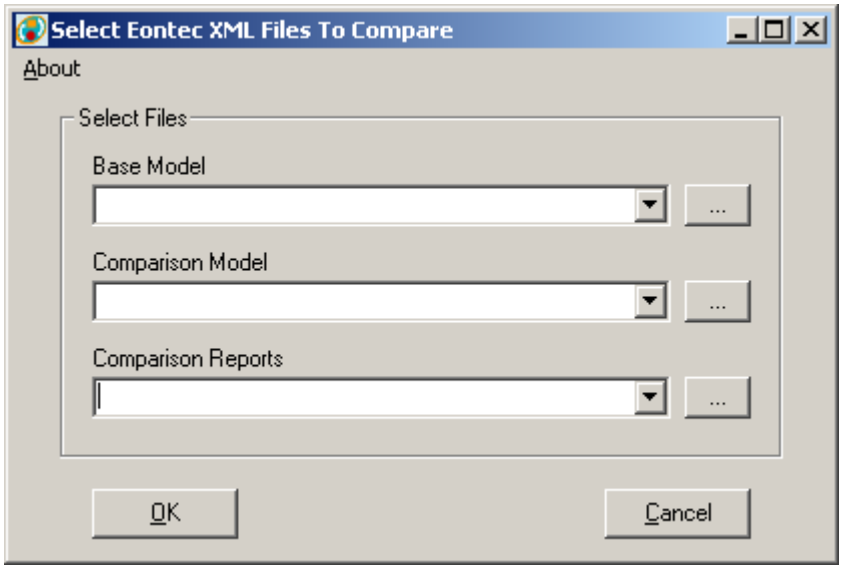
It produces 6 reports in HTML format, one for each of the following:

- Banking Objects
- Banking Processes
- Parameter Objects
- Common Validations
- Validator Lengths
- Constants

These comparison reports can be used to identify changes to a model during the design, development and testing phases of a project. The details of these reports are listed in the Report Contents section.

11.2 Using the Comparison Tool

The main screen of the Comparison Tool appears as follows:



The following steps should be taken to compare two Models:

1. Click on the browse button beside the Base Model list – a file chooser dialog is then launched - select the base model XML, i.e. the old version of the model that you want to compare against

2. Click on the browse button beside the Comparison Model list – a file chooser dialog is then launched - select the comparison model XML, i.e. the newest version of the model to compare against
3. Click on the browse button beside the Comparison Reports list to select the location that the comparison report files will be output to
4. Click on the “OK” button to generate the Comparison Reports.
5. When the reports have been generated a message will be displayed indicating where the comparison reports have been output to. It may take a few minutes for the generation process to complete depending on the size of the models being compared

Note: The XML files used as input into this tool must be valid Model XML files. They should be exported from a Rose Model that adheres to all Design Standards as specified in the Automated Methodology. The XML export tool can be found in the Utilities section of the Design Tools.

11.2.1 Error Messages

The following errors can be encountered when using the Comparison Tool

11.2.1.1 XML Parsing Error – An invalid character was found in text content

This error indicates that an invalid XML character has been found in one of the selected XML files. Ensure that the Model Validator (found in the Utilities section of the Design Tools) has been run and correct any invalid XML characters which are flagged in the Model Validator report. The error message above indicates the line number that the invalid character was found at. This line of the XML file can be inspected in a text editor to help to identify where the problem is located. However it is recommended that the actual Rational Rose Model be updated to correct the problem rather than the XML file. If the XML file is edited manually the changes will not be reflected in the Rose Model and the problem will occur again the next time XML is exported from the model.

11.2.1.2 XML Parsing Error – The system cannot locate the resource specified

This error message indicates that the Model.dtd is missing. The Retail Fiance Tools `Model.dtd` is installed in the same default location. The Comparison Tool cannot parse a Siebel Model XML file if the dtd cannot be found on the system. However an XML file generated on another machine that uses an older version of the Design Tools may have a reference to a different folder location in its DOCTYPE (located on the second line of the file). To correct this problem the XML file must be edited to change the location of the EontecModel.dtd to a valid location on the users system.

Note: Great care should be taken when manually editing the XML files. If a tag is left open or an invalid character inserted by accident the tool will not be able to read the file.

11.2.1.3 XML Parsing Error – The base XML file selected is not of doctype EontecModel.

This error indicates that an XML file was selected which wasn't of DOCTYPE EontecModel. This means that the file chosen was not a valid Siebel Model XML file and hadn't been exported from the Design Tools.

11.2.1.4 XML Parsing Error – A duplicate package name has been found in the base XML.

This error indicates that the XML file contains more than one object with the same package name. According to Automated Methodology standards all classes should have unique package names. The duplicate package names should be removed from the design before the comparison tool can be run again.

11.2.2 Report Contents

The following are the Generated Reports (HTML files) that are produced by the Model Comparison Tool:

- Banking Objects Report (BankingObjects.html)
- Banking Processes (BankingProcesses.html)
- Common Validations (CommonValidations.html)
- Constants (Constants.html)
- Parameter Objects (ParameterObjects.html)
- Validator Lengths

Note: If there are no differences found for a particular report only the initial section (see below) is displayed. Underneath this the message "No Differences Found" is displayed.

11.2.2.1 Initial Section

All reports contain a header area containing the following information:

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

11.2.2.2 The Summary Report section

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

11.2.2.3 The Detailed Report Section

11.2.2.3.1 Banking Object Report (BankingObjects.html)

Added Objects - The name and package name of Banking Objects that have been added to the model are listed here. Note: The properties, attributes and methods of objects that have been added are not listed in the report.

Removed Objects - The name and package name of Banking Objects that have been removed from the

model are listed here. Note: The properties, attributes and methods of objects that have been added or removed are not listed in the report.

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 table.

The following sub-sections can be found in this section of the report:

- Modified Object Details - Any properties of the Banking Object that have been modified are listed here.
- Modified Attributes - Any attributes that have been modified are listed here. 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 are listed here
- Removed Attributes – Any attributes that have been removed from the object are listed here
- Modified Methods – Any methods that have been modified are listed here. It is important to remember that it is possible for the same method name to exist with many signatures. Therefore, if the signature of an existing method on a Class is modified through 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 are listed here.
- Removed Methods – The name and signature of any methods that have been removed from the object are listed here.

11.2.2.3.2 Banking Process Report (BankingProcesses.html)

Added Sessions- The name and package name of Sessions that have been added to the model are listed here.

Removed Sessions - The name and package name of Sessions that have been removed from the model are listed here.

Note: The properties and processes of sessions that have been added or removed are not listed in the report.

Modified Sessions

Each modified session is represented as a table in the report. The session's name and package name are listed at the top of the table. The modifications to the session are listed in the table.

The following sub-sections can be found in this section of the report:

- Modified Object Details - Any properties of the Session that have been modified are listed here.

- Modified Processes – Any processes that have been modified are listed here. It is important to remember that it is possible for the same process name to exist with many signatures. Therefore, if the signature of an existing process on a session is modified 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 are listed here.
- Removed Processes – The name and signature of any processes that have been removed from the object are listed here.

11.2.2.3.3 Common Validations Report (CommonValidations.html)

Note: This report only outlines changes to Common Validations. Customized validations are stored on an attribute. Therefore if an attribute's validation is customized, this will appear on the BankingObject comparison report.

Added Objects- The name and package name of CommonValidations objects that have been added to the model are listed here. Note: The Validation methods of Common Validations objects that have been added are not listed in the report

Removed Objects - The name and package name of CommonValidations objects that have been removed from the model are listed here. Note: The Validation methods of Common Validations objects that have been removed are not listed in the report

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 table.

The following sub-sections can be found in this section of the report:

- Modified Validations– Any validations that have been modified are listed here.
- Added Validations – The name of any validations that have been added to the object are listed here.
- Removed Validations – The name of any validations that have been removed from the object are listed here.

11.2.2.3.4 Constants Report (Constants.html)

Added Constants Classes- The name and package name of CommonValidations objects that have been added to the model are listed here.

Removed Constants Classes - The name and package name of CommonValidations objects that have been removed from the model are listed here.

Note: There should only be one Constants class in the model (as per Automated Methodology standards). If the package name of this class is changed the report will indicate that the class has been removed and a new one added.

Added Constants

Any Constants that have been added to the Constants Class are listed here.

Removed Constants

Any Constants that have been removed from the Constants Class are listed here.

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 inside this table.

The following sub-sections can be found in this section of the report:

- Modified Constant Details – The only constant detail that can change is the data type. If this changes the details are listed here.
- Added Values – The name of any constant values that have been added to the constant are listed here.
- Removed Values – The name of any constant values that have been removed from the constant are listed here.

Note: The term Constant above shouldn't be confused with the term Constants Class. A Constant represents a value or list of values. An example of a Constant would be `ACCOUNT_TYPE`. The `ACCOUNT_TYPE` constant could contain Constant Values of "Current Account" and "Deposit Account". The Constants Class is the Class that these constants are stored on in the model.

11.2.2.3.5 Parameter Object Report (ParameterObjects.html)

Added Objects - The name and package name of Parameter Objects that have been added to the model are listed here.

Removed Objects - The name and package name of Parameter Objects that have been removed from the model are listed here.

Note: The properties and attributes of objects that have been added or removed are not listed in the report.

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 inside this table.

The following sub-sections can be found in this section of the report:

- Modified Object Details - Any properties of the Banking Object that have been modified are listed here.
- Modified Attributes - Any attributes that have been modified are listed here. 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 are listed here
- Removed Attributes – Any attributes that have been removed from the object are listed here

11.2.2.3.6 Validator Lengths Report (ValidatorLengths.html)

Note: Validator Lengths are stored on the Constants Class in Siebel XML representation of the design model. If the package name of the Constants Class changes then it is considered to have been removed and a new one added. If this has happened then the Comparison Tool doesn't examine the Validator Lengths for differences and only the initial section of the report is generated. Underneath this the message "The Constants Class has been removed or renamed" is displayed.

Added Validator Lengths

Any Validator Lengths that have been added to the model are listed here.

Removed Validator Lengths

Any Validator Lengths that have been removed from the model are listed here.

Modified Validator Lengths

The name and details of each modified Validator Length are listed is this section.

Note: The only Validator Length detail that can change is the length.