Code Engineering Using UML Models

Enterprise Architect is an intuitive, flexible and powerful UML analysis and design tool for building robust and maintainable software.

This booklet describes the code engineering facilities of Enterprise Architect.

Enterprise Architect - Code Engineering Using UML Models

Printed: May 2010

Publisher

Sparx Systems

Managing Editor

Geoffrey Sparks

Technical Editors

Geoffrey Sparks
Simon McNeilly
Vimal Kumar
Henk Dekker

Special thanks to:

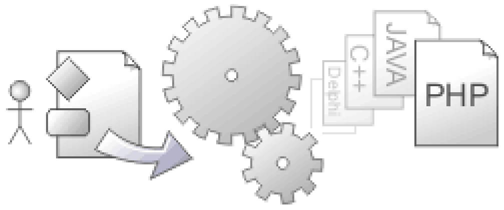
All the people who have contributed suggestions, examples, bug reports and assistance in the development of Enterprise Architect. The task of developing and maintaining this tool has been greatly enhanced by their contribution.

Table of Contents

Foreword

This user guide describes the code engineering facilities of Enterprise Architect.

1 Code Engineering



Code Engineering is a process that includes **automated code generation**, **reverse engineering** of source code and **synchronization** between the source code and model.

Enterprise Architect also enables you to rapidly model, generate - or *forward engineer* - and reverse engineer:

- XML Technologies 88, namely XML Schema (XSD) and Web Service Definition Language (WSDL)
- Database schema 120, keys, triggers, constraints, RI and other relational database features, for and from a range of database products.

Code Engineering is available in the Professional, Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect.

Code Generation

Enterprise Architect enables you to generate source code 12 from UML model elements, creating a source code equivalent of the Class or Interface element for future elaboration and compilation. In particular you can generate C, C++, C#, Delphi, Java, PHP, Python, ActionScript, Visual Basic and VB.NET 42 source code. The source code generated includes Class definitions, variables and function stubs for each attribute and method in the UML Class. You can use the *Source Code Viewer* to view any source code you are opening (see the *Dockable Windows* section of *Using Enterprise Architect - UML Modeling Tool*).

Note:

You view source code for an element by selecting it and pressing either **[Ctrl]+[E]** or **[F12]**. If the element does not have a generation file (that is, code has not been or cannot be generated, such as for a Use Case), Enterprise Architect checks whether the element has a link to either an operation or an attribute of another element. If such a link exists, and that other element has source code, the code for that element displays.

You can also generate code from three UML behavioral modeling 19 paradigms:

- State Machine diagrams
- Interaction diagrams
- Activity diagrams.

The *Code Template Framework (CTF)* 61 enables you to customize the way Enterprise Architect generates source code. It also enables you to generate languages that Enterprise Architect does not specifically support, by helping you define the appropriate code generation templates for that language (this is discussed in *SDK for Enterprise Architect)*.

You can integrate the facilities of Enterprise Architect with other development environments. The MDG Integration for Eclipse and MDG Integration for Visual Studio 10 are standalone products that provide an enhanced code engineering functionality between Enterprise Architect and the development environments.

Reverse Engineering

Reverse Engineering 4 is the import of existing source code into model elements, mapping the source code structures onto their UML representations. This enables you to examine legacy code and the functionality of code libraries for reuse, or to bring the UML model up to date with the code. You can reverse engineer in the same languages as you perform code generation with Enterprise Architect.

Enterprise Architect is also able to reverse engineer binary files, namely Java .jar files and .NET PE files.

Note:

Reverse Engineering of other languages including CORBA IDL is also currently available through the use of the MDG Technologies. See www.sparxsystems.com/resources/mdg_tech/.

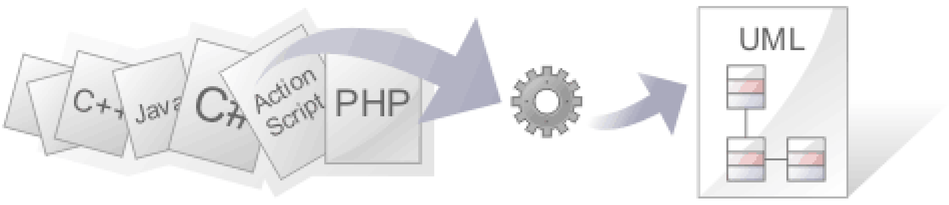
Synchronization

Synchronization 10 is when changes in the model are exported to the source code and changes to source code are imported into the model. This enables you to keep your model and source up to date as the project develops.

Round-Trip Engineering

Round trip engineering is a combination of reverse and forward generation of code and includes synchronization between the source code and the model in all but the most trivial of code engineering projects. In order to get the most out of round trip engineering in Enterprise Architect, you should be familiar with the modeling conventions 68 used when generating and reverse engineering the languages you use.

1.1 Reverse Engineering



Reverse Engineering in Enterprise Architect enables you to import existing source code from a variety of code languages into a UML model. Existing source code structures are mapped into their UML representations, for example a Java Class is mapped into a UML Class element with the variables being defined as attributes, methods are modeled as operations and the interactions between the Java Classes being displayed in the UML model Class diagram with the appropriate connectors.

Reverse Engineering enables users to examine legacy code and examine the functionality of code libraries for reuse or to bring the UML model up to date with the code that has been developed as part of a process called *synchronization*. Examining the code in a UML model enables user to identify the critical modules contained the code, enabling a starting point for understanding of the business and system requirements of the pre-existing system and to enable the developers to gain a better overall understanding of the source code.

To begin the process of importing existing code into Enterprise Architect, an existing source of code must be imported into Enterprise Architect 5, which can be a single directory or a directory structure 8. Several options are available when performing the reverse engineering process. The *Source Code Engineering Options* 32 topic contains several options that affect the reverse engineering process. These include:

- If comments are reverse engineered into notes fields, and how they are formatted if they are
- How property methods are recognized
- If dependencies should be created for operation return and parameter types.

It is important to note that when a legacy system is not well designed, simply importing the source into Enterprise Architect does not create an easily understandable UML model. When working with a legacy system that is poorly designed it is useful to break down the code into manageable components by examining the code elements individually. This can be achieved by importing a specific Class of interest into a diagram and then inserting the related elements (see *UML Modeling with Enterprise Architect – UML Modeling Tool*) at one level to determine immediate relationship to other Classes. From this point it is possible to create Use Cases that identify the interaction between the legacy Classes, enabling an overview of the legacy system's operation.

Copyright ownership is an important issue to take into account when undertaking the process of reverse engineering. In some cases, software might have specific limitations that prohibit the process of reverse engineering. It is important that a user address the issue of copyright before beginning the process of reverse engineering code. Situations that typically lend themselves to reverse engineering source code include source code that:

- You have already developed
- Is part of a third-party library that you have obtained permission to use
- Is part of a framework that your organization uses
- Is being developed on a daily basis by your developers.

Enterprise Architect currently supports reverse engineering in the following programming languages:

- ActionScript 69
- Ada 2005 69 (Systems Engineering and Ultimate editions)
- C 71
- C# 73
- C++ 75
- Delphi 78
- Java 79
- PHP 80
- Python 81
- SystemC 81 (Systems Engineering and Ultimate editions)
- Verilog 84 (Systems Engineering and Ultimate editions)

- VHDL 85 (Systems Engineering and Ultimate editions)
- Visual Basic 87
- Visual Basic .NET 83

Enterprise Architect is also able to reverse engineer certain types of binary files: Java .jar files and .NET PE files. See Import Binary Module 9 for more information.

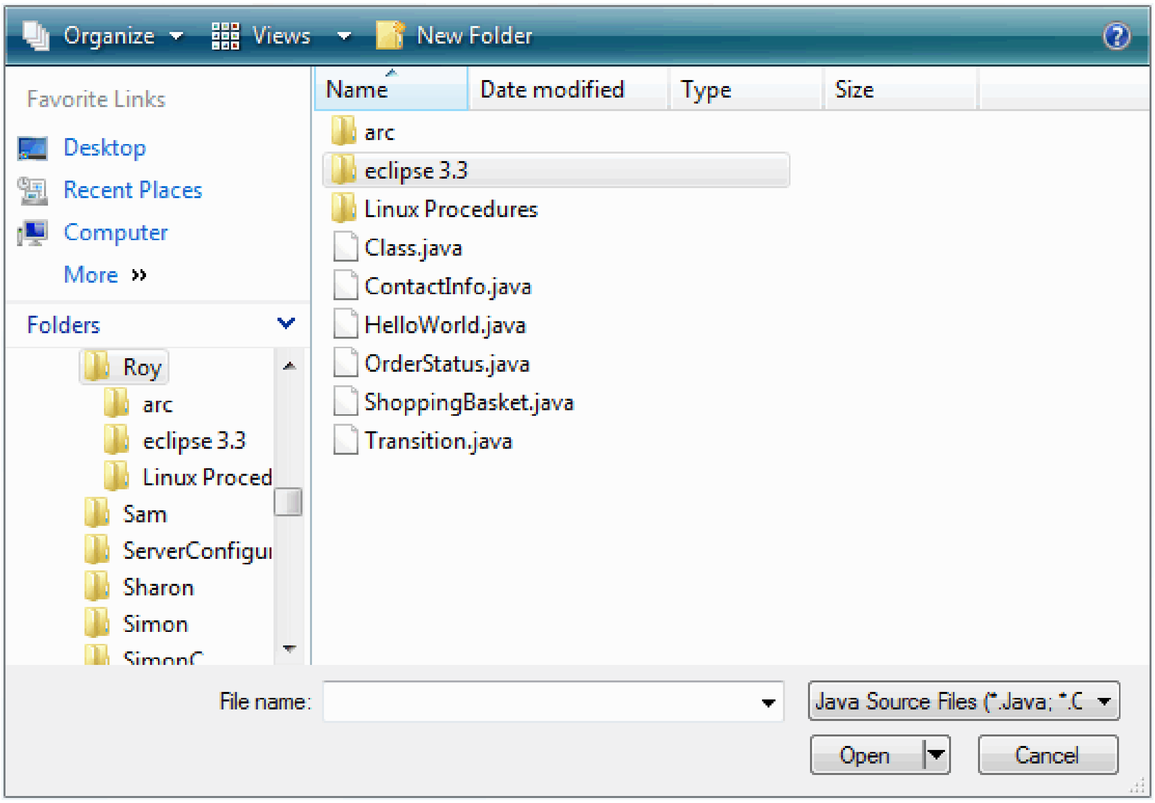
Notes:

- Reverse Engineering of other languages is currently available through the use of MDG Technologies from www.sparxsystems.com/resources/mdg_tech/.
- In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Reverse Engineer From DDL And Source Code** permission to reverse engineer source code and synchronize model elements against code. See *User Security in UML Models.*

1.1.1 Import Source Code

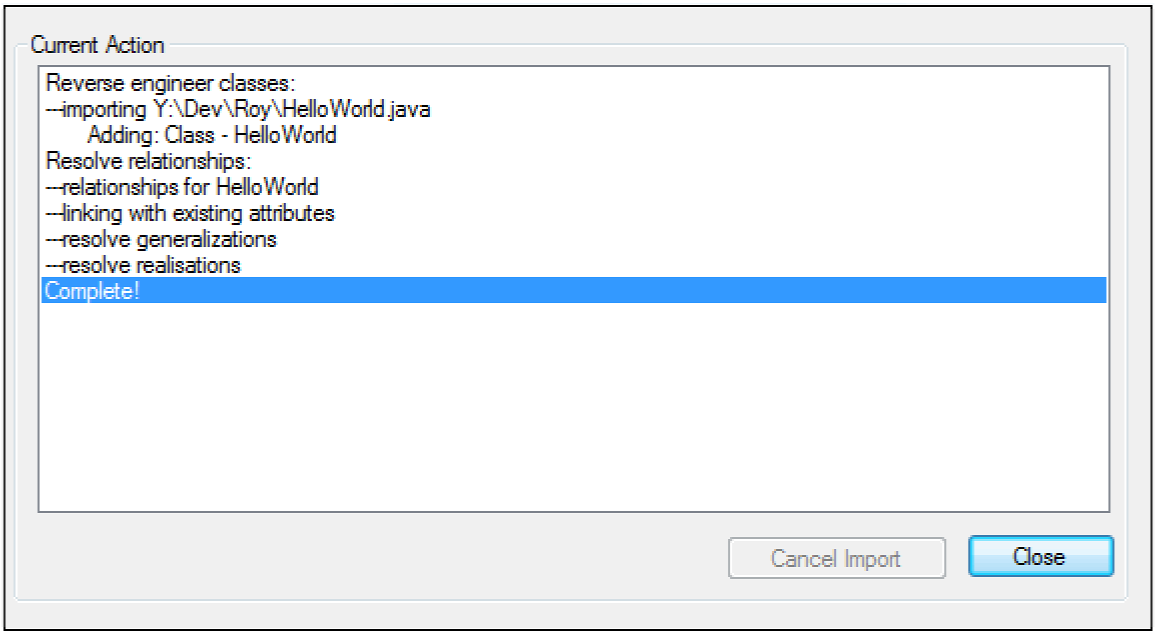
To import source code (reverse engineer) follow the steps below:

1. In the Project Browser, select (or add) a diagram into which to import the Classes.
2. Right-click on the diagram background to open the context menu and either:
   - Select the language to import from the **Import from source file(s)** submenu, or
   - Click on the **Import Language** drop-down arrow in the Code Generation toolbar and select the **Import | Import xxx files** menu option, where *xxx* represents the language to import.
3. From the file browser that appears, select one or more source code files 6 to import.



4. Click on the **Open** button to start the import process.

As the import proceeds, Enterprise Architect provides progress information. When all files are imported, Enterprise Architect makes a second pass to resolve associations and inheritance relationships between the imported Classes.



```
Reverse engineer classes:
--importing Y:\Dev\Roy\HelloWorld.java
      Adding: Class - HelloWorld
Resolve relationships:
--relationships for HelloWorld
--linking with existing attributes
--resolve generalizations
--resolve realisations
Complete!
```

1.1.2 Notes on Source Code Import

Enterprise Architect enables you to import code 5 into your project, in the following programming languages:

- ActionScript 6
- C 6
- C# 7
- C++ 6
- Delphi 7
- Java 7
- PHP 7
- Python 7
- Visual Basic 7
- Visual Basic .NET 7

Enterprise Architect supports most constructs and keywords for each coding language.

If there is a particular feature you require support for that you feel is missing, please contact Sparx Systems.

You must select the appropriate type of source file for the language, as the source code to import.

ActionScript

Appropriate type of source file: **.as**.

C

Appropriate type of source file: **.h** header files and/or **.c** files.

When you select a header file Enterprise Architect automatically searches for the corresponding **.c** implementation file to import based on the options for extension and search path specified in the C options 44 .

Enterprise Architect does not expand macros that have been used, these must be added into the internal list of Language Macros 39.

C++

Appropriate type of source file: **.h** header file.

Enterprise Architect automatically searches for the **.cpp** implementation file based on the extension and search path set in the C++ options 46. When it finds the implementation file it can use it to resolve parameter names and method notes as necessary.

When importing C++ source code, Enterprise Architect ignores function pointer declarations. To import them into your model you could create a *typedef* to define a function pointer type, then declare function pointers using that type. Function pointers declared in this way are imported as attributes of the function pointer type.

Enterprise Architect does not expand macros that have been used; these must be added into the internal list of Language Macros 39.

C#

Appropriate type of source file: **.cs**.

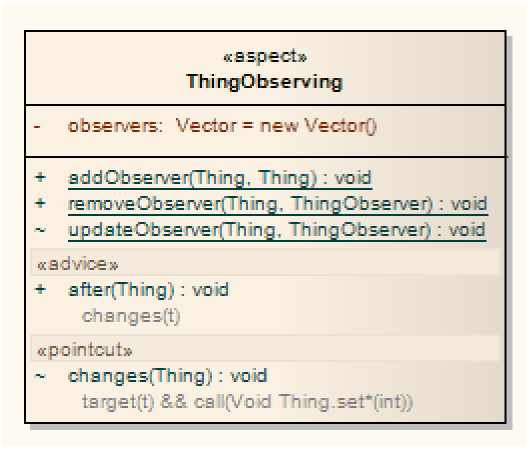
Delphi

Appropriate type of source file: **.pas**.

Java

Appropriate type of source file: **.java**.

Enterprise Architect supports the AspectJ language extensions.


«aspect»
ThingObserving
- observers: Vector = new Vector()
+ addObserver(Thing, Thing) : void
+ removeObserver(Thing, ThingObserver) : void
~ updateObserver(Thing, ThingObserver) : void
«advice»
+ after(Thing) : void
changes(t)
«pointcut»
~ changes(Thing) : void
target(t) && call(Void Thing.set*(int))

Aspects are modeled using Classes with the stereotype *aspect*. These aspects can then contain attributes and methods as for a normal Class. If an *intertype* attribute or operation is required, you can add a tag *className* with the value being the name of the Class it belongs to.

Pointcuts are defined as operations with the stereotype of *pointcut*. These can occur in any Java Class, Interface or aspect. The details of the pointcut are included in the **behavior** field of the method.

Advice is defined as an operation with the stereotype *advice*. The pointcut this advice operates on is in the **behavior** field and acts as part of the method's unique signature. After advice can also have one of the Tagged Values *returning* or *throwing*.

PHP

Appropriate type of source file: **.php**, **.php4**, or **.inc**.

Python

Appropriate type of source file: **.py**.

Visual Basic

Appropriate type of source file: **.cls** Class file.

Visual Basic .NET

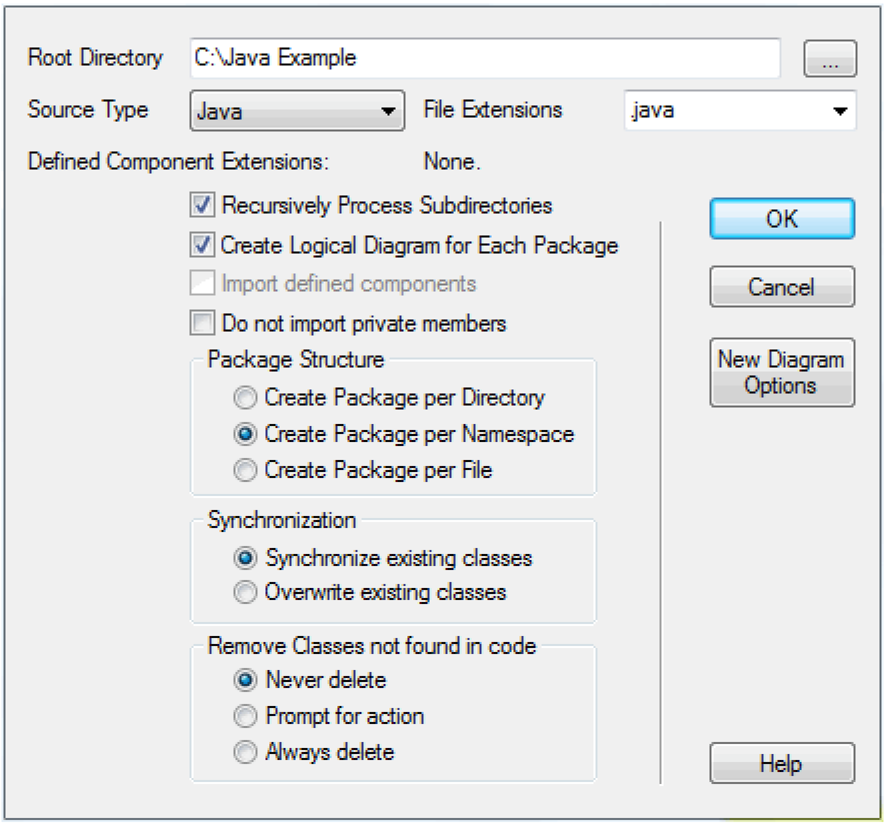
Appropriate type of source file: **.vb** Class file.

1.1.3 Import a Directory Structure

You can import from all source files in a complete directory structure. This process enables you to import or synchronize multiple files in a directory tree in one pass. Enterprise Architect creates the necessary packages and diagrams during the import process.

To import a directory structure, follow the steps below:

1. In the Project Browser, right-click on the target package for the import.
2. From the context menu, select the **Code Engineering | Import Source Directory** menu option. The Import Source Directory dialog displays.



Root Directory: C:\Java Example

Source Type: Java

File Extensions: .java

Defined Component Extensions: None.

☑ Recursively Process Subdirectories
☑ Create Logical Diagram for Each Package
☐ Import defined components
☐ Do not import private members

Package Structure
○ Create Package per Directory
● Create Package per Namespace
○ Create Package per File

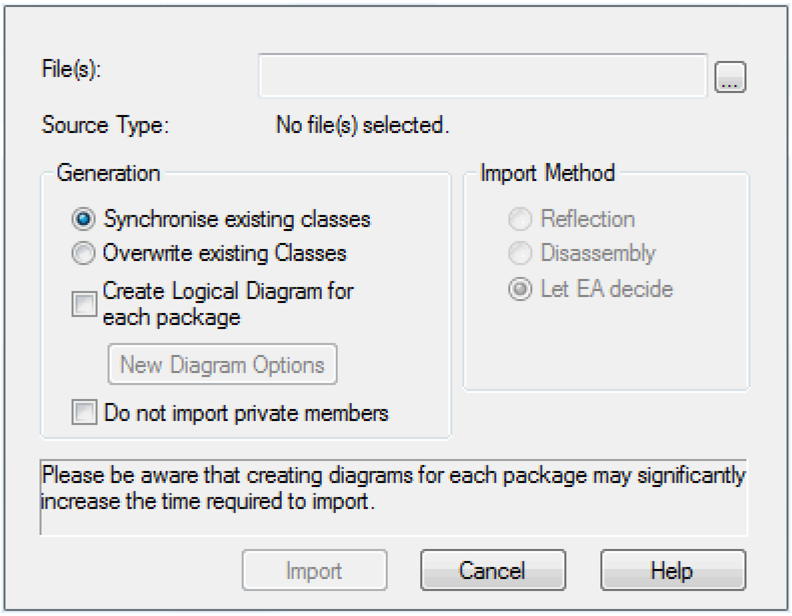
Synchronization
● Synchronize existing classes
○ Overwrite existing classes

Remove Classes not found in code
● Never delete
○ Prompt for action
○ Always delete

3. Select the options you require. You can configure:
   - The source directory
   - The source type
   - The file extensions to look at
   - Whether to recurse sub directories
   - Whether to create a diagram for each package
   - Whether to import additional files as described in the Import Component Types dialog
   - Whether to exclude private members from libraries being imported from the model
   - Whether to create a package for every directory, namespace or file; this might be restricted depending on the source type selected
   - Whether to Synchronize or Overwrite existing Classes when found (if a model Class is found matching the one in code, **Synchronize** *updates* the model Class to include the details from the one in code, which preserves information not represented in code such as the location of Classes in diagrams; **Overwrite** *deletes* the model Class and generates a new one from code, which deletes and does not replace the additional information)
   - How to handle Classes not found during the import (**Prompt for action** enables you to review Classes individually 10)
   - What is shown on diagrams created by the import.
4. Click on the **OK** button to start.

1.1.4 Import Binary Module

Enterprise Architect enables you to reverse-engineer certain types of binary modules. To import a binary module, right-click on the target package in the Project Browser and select the **Code Engineering | Import Binary Module** context menu option.



Source Type: No file(s) selected.

Generation: (● Synchronise existing classes) ○ Overwrite existing Classes; ☐ Create Logical Diagram for each package; ☐ Do not import private members

Import Method: ○ Reflection ○ Disassembly (● Let EA decide)

Please be aware that creating diagrams for each package may significantly increase the time required to import.

Currently the permitted types are as follows:

- Java Archive (.jar)
- .Net PE file (.exe, .dll); native Windows DLL and EXE files are not supported, only PE files containing .NET assembly data
- Intermediate Language file (.il).

Enterprise Architect creates the necessary packages and diagrams during the import process. Selecting the **Do not import private members** checkbox excludes private members from libraries from being imported into the model.

When importing .Net files,you can import via reflection or via disassembly, or let Enterprise Architect decide the best method - this might result in both types being used. The reflection-based importer relies on a .Net program, and requires the .Net runtime environment to be installed. The disassembler-based importer relies on a native Windows program called *Ildasm.exe,* which is a tool provided with the MS .Net SDK. The SDK can be downloaded from the Microsoft website.

A choice of import methods is available because some files are not compatible with reflection (such as *mscorlib.dll*) and can only be opened using the disassembler. However, the reflection-based importer is generally much faster.

You can also configure:

- Whether to **Synchronize** or **Overwrite** existing Classes when found (if a model Class is found matching the one in the file, **Synchronize** updates the model Class to include the details from the one in the file, which preserves information not represented in the file such as the location of Classes in diagrams; **Overwrite** deletes the model Class and generates a new one from the file, which deletes and does not replace the additional information)
- Whether to create a diagram for each package
- What is shown on diagrams created by the import.

1.1.5 MDG Integration and Code Engineering

MDG Integration for Eclipse and MDG Integration for Visual Studio are standalone products that provide an enhanced code engineering functionality between Enterprise Architect and the development environments.

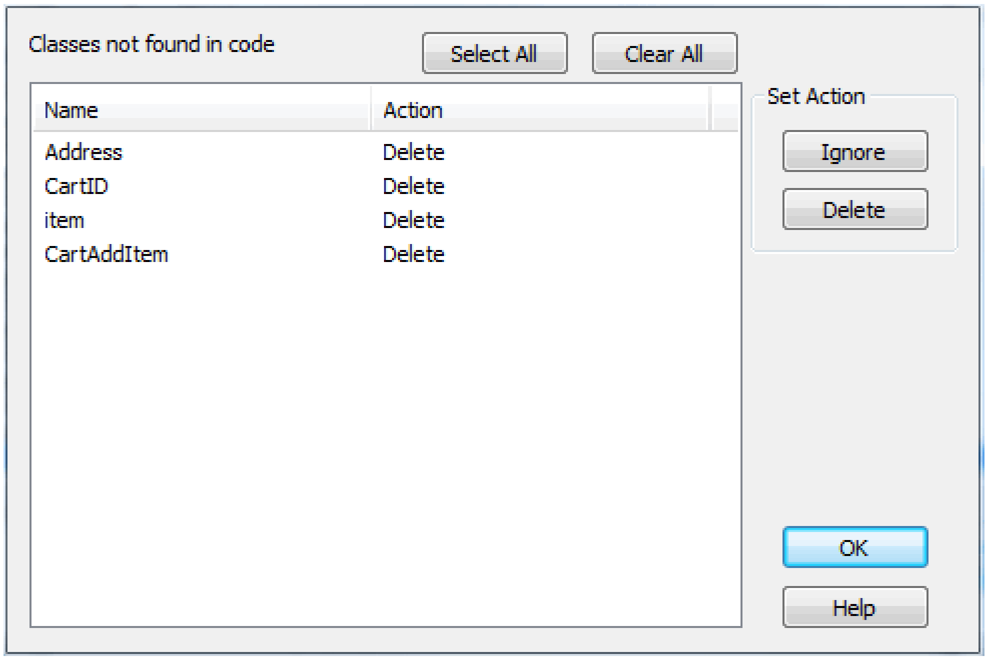
The MDG Integration programs provide a lightweight bridge between Enterprise Architect and the development environment, offering enhanced code generation, reverse engineering and synchronization between code and the UML model. Merging changes can be achieved with minimal effort, and navigation between model and source code is significantly enhanced.

A trial version of MDG Integration for Eclipse can be downloaded from www.sparxsystems.com/products/mdg/int/eclipse/index.html and MDG Integration for Visual Studio can be downloaded from www.sparxsystems.com/products/mdg/int/vs/index.html.

1.1.6 Classes Not Found During Import

When reverse synchronizing from your code, there are times when some Classes might be deliberately removed from your source code. Enterprise Architect's import source directory functionality keeps track of the Classes it expects to synchronize with and, on the Import Directory Structure dialog, provides options for how to handle the Classes that weren't found. You can select the appropriate action so that, at the end of the import, Enterprise Architect either ignores the missing Classes, automatically deletes them or prompts you to handle them.

If you select the **Prompt For Action** 8 radio button on the Import Directory Structure dialog, to manually review missing Classes, the following dialog displays:



Name	Action
Address	Delete
CartID	Delete
item	Delete
CartAddItem	Delete

By default, all Classes are marked for deletion. To keep one or more Classes, select them and click on the **Ignore** button.

1.1.7 Synchronize Model and Code

In addition to generating and importing code, Enterprise Architect provides the option to synchronize the model and source code, creating a model that represents the latest changes in the source code and vice versa. You can use either the model as the source, or the code as the source.

For example: you generated some source code, but made subsequent changes to the model. When you generate code again, Enterprise Architect *adds* any new attributes or methods to the existing source code, leaving intact what already exists. This means developers can work on the source code and then generate *additional* methods as required from the model, without having their code overwritten or destroyed.

Note:

Code synchronization does not *change* method bodies. Behavioral code generation 19 only works when generating the entire file.

Similarly, you might have made changes to a source code file, but the model has detailed notes and characteristics you do not want to lose. By synchronizing from the source code into the model, you import additional attributes and methods but do not change other model elements.

Using the two synchronization methods above, it is simple to keep source code and model elements up to date and synchronized.

Note:

In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Generate Source Code and DDL** permission to synchronize source code with model elements. See *User Security in UML Models.*

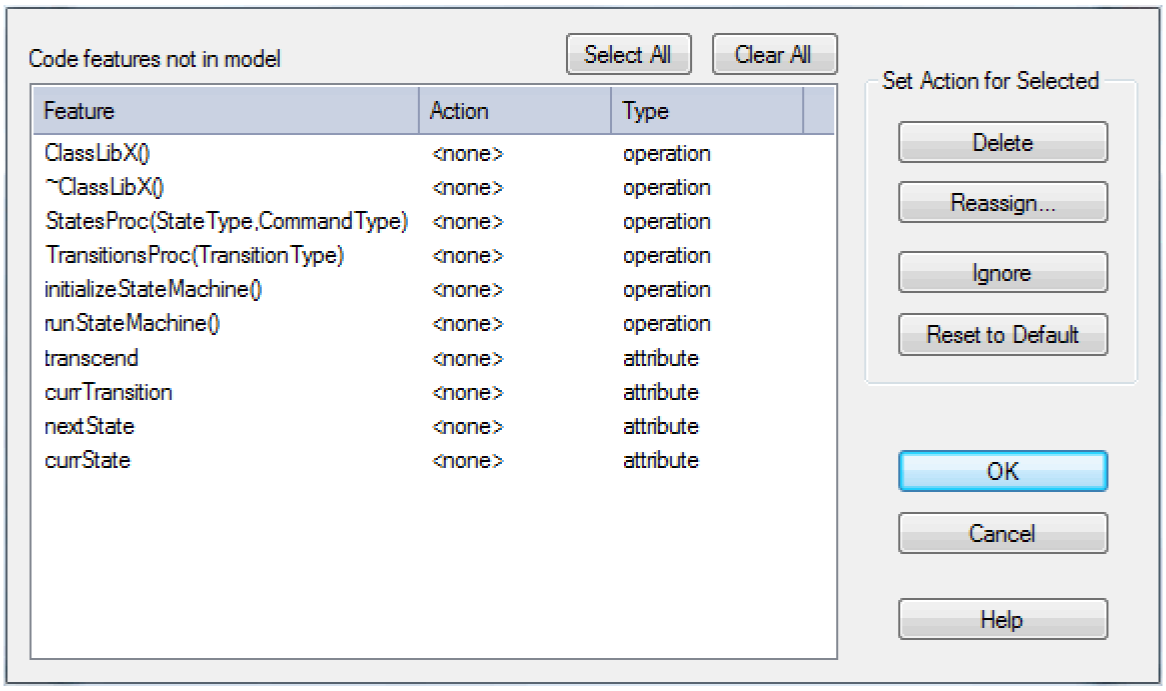
Synchronize Classes on Forward Generation

When there are features present in the code but not in the model you can use the following buttons during forward synchronization:

Note:

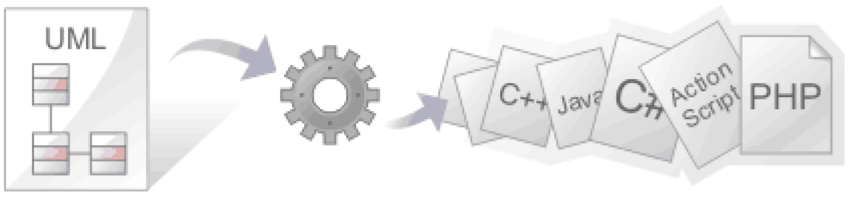
These buttons are only available when the **On forward synch, prompt to delete code features not in model** checkbox is selected in the Options - Attributes and Operations 36 dialog.

- **Delete**: when you click on this button the selected code features are removed from the code.
- **Reassign**: when you click on this button the code elements are reassigned to elements in the model (this is only possible when an appropriate model element is present that is not already defined in the code).
- **Ignore**: when you click on this button the code elements not present in the model are ignored completely.
- **Reset to Default**: when you click on this button the settings for synchronizing during forward generation are set to Ignore, meaning that the elements present in the code but not in the model are ignored completely.



Feature	Action	Type
ClassLibX()	<none>	operation
~ClassLibX()	<none>	operation
StatesProc(StateType,CommandType)	<none>	operation
TransitionsProc(TransitionType)	<none>	operation
initializeStateMachine()	<none>	operation
runStateMachine()	<none>	operation
transcend	<none>	attribute
currTransition	<none>	attribute
nextState	<none>	attribute
currState	<none>	attribute

1.2 Generate Source Code



Generating source code (forward engineering) takes the UML Class or Interface model elements and creates a source code equivalent for future elaboration and compilation. By forward engineering code from the model, the mundane work involved with having to key in Classes and attributes and methods is avoided, and symmetry between model and code is ensured.

Code is generated from *Class* or *Interface* model elements, so you must create the required Class and Interface elements to generate from. Add attributes (which become variables) and operations (which become methods).

Before you generate code, you should ensure the default settings for code generation match your requirements. The default generation settings are located in the Source Code Engineering page of the Options dialog (select the **Tools | Options | Source Code Engineering** menu option). Set up the defaults to match your required language and preferences. Preferences that you can define include default constructors and destructors, methods for interfaces and the Unicode options for created languages. Languages such as Java support namespaces 17 and can be configured to specify a namespace root. In addition to the default settings for generating code, Enterprise Architect supports the following code languages with their own specific code generation options:

- ActionScript 43
- C 44
- C# 45 (for both .NET 1.1 and .NET 2.0)
- C++ 46 (standard, plus .NET managed C++ extensions)
- Delphi 47
- Java 51 (including Java 1.5, Aspects and Generics)
- PHP 51
- Python 52
- Visual Basic 57
- Visual Basic .NET 54

The Code Template Framework (CTF) 61 enables you to customize the way Enterprise Architect generates source code and also enables generation of languages that are not specifically supported by Enterprise Architect.

Before generating code, you should also familiarize yourself with the way Enterprise Architect handles local path names. Local path names enable you to substitute tags for directory names (for example %SRC% = C:\Source).

When you have completed the design of your Classes, you can generate source code.

Note:

In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Generate Source Code and DDL** permission to generate source code. See *User Security in UML Models.*

Use Live Code Generation

On the Package Build Scripts dialog, you have the option to update your source code instantly as you make changes to your model. (See the *Setup for Execution Analysis* topic in *Visual Execution Analyzer in Enterprise Architect.*)

Tasks

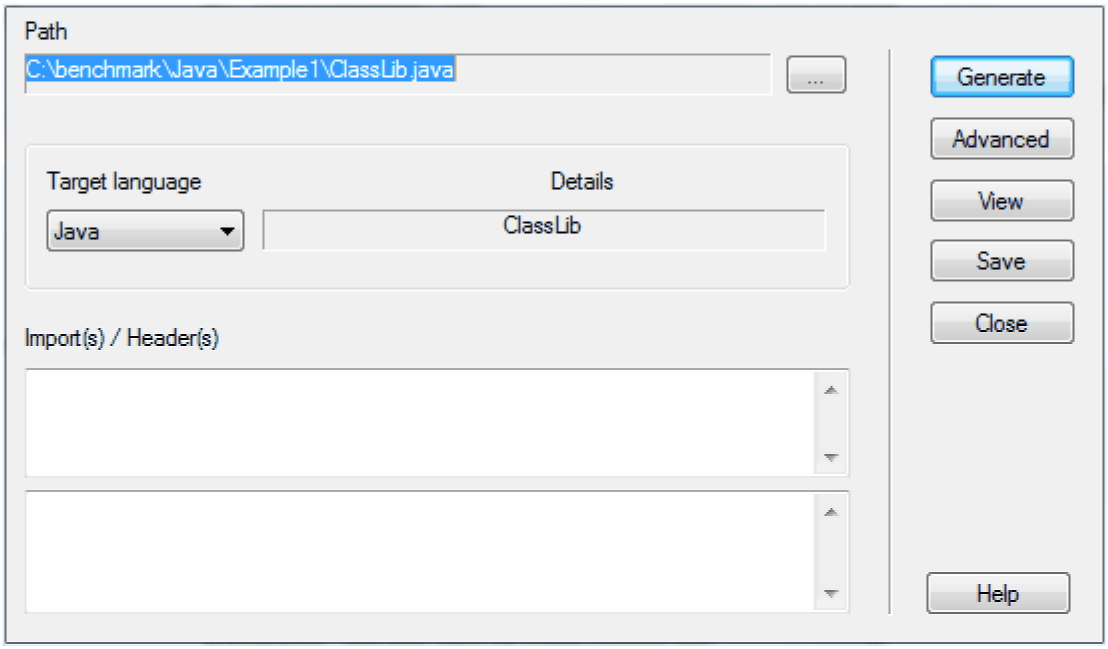
When you generate code, you perform one or more of the following tasks:

1.2.1 Generate a Single Class

To generate code for a single Class, first ensure the design of the model element (Class or Interface) is complete. Also ensure you have added Inheritance connectors to parents and associations to other Classes that are used. Also add Inheritance connectors to Interfaces that your Class implements; Enterprise Architect offers the option to generate function stubs for all interface methods that a Class implements. Once the design is satisfactory, follow the steps below.

Generate Code for a Single Class

1. Open the diagram containing the Class or Interface for which to generate code.
2. Right-click on the required Class or Interface to display the context menu and select the **Generate Code** menu option, or press **[F11]**. The Generate Code dialog displays, which enables you to control how and where your source code is generated.



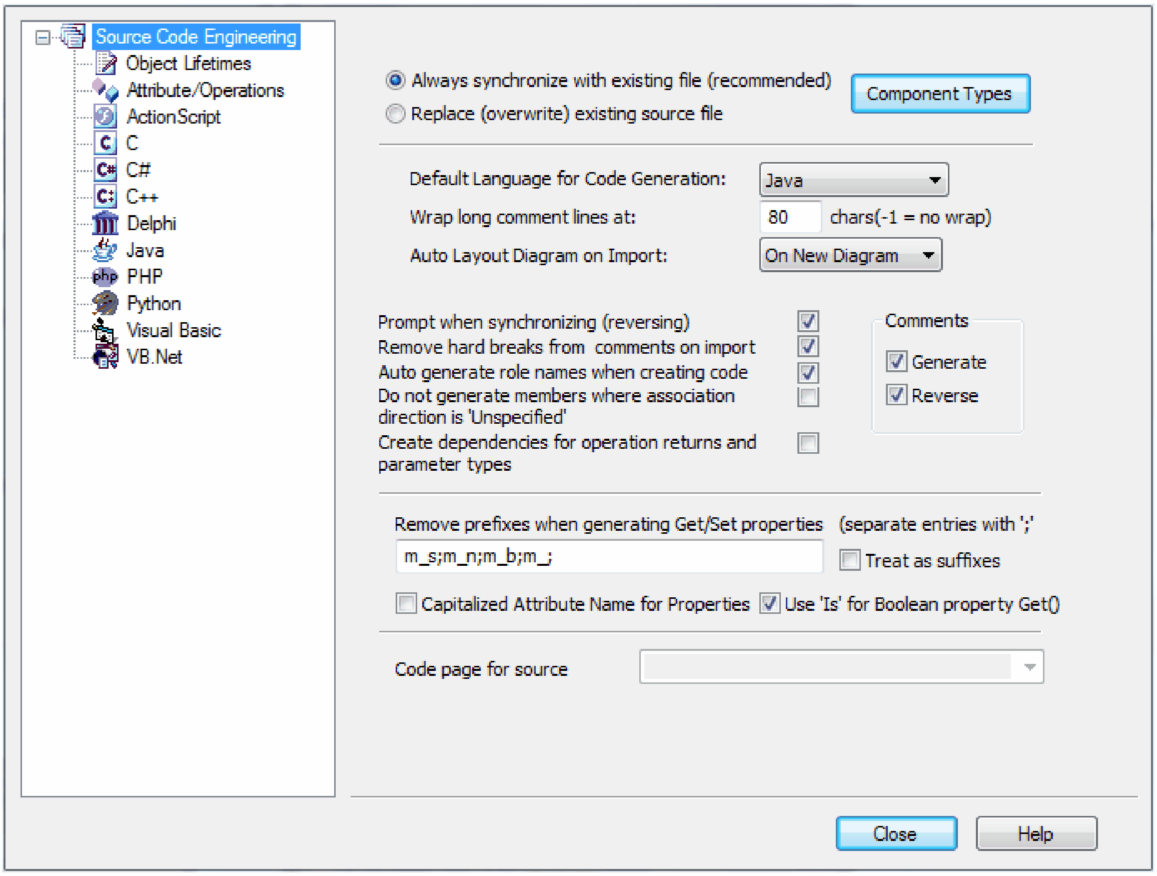
Path: C:\benchmark\Java\Example1\ClassLib.java

Target language: Java

Details: ClassLib

Import(s) / Header(s)

3. On the **Path** field, click on **[...]** (Browse) and select a path name for your source code to be generated to.
4. In the **Target Language** field, click on the drop-down arrow and select the language to generate; this becomes the permanent option for that Class, so change it back if you are only doing one pass in another language.
5. Click on the **Advanced** button. The Object Options dialog displays.



[✓ Always synchronize with existing file (recommended)] Replace (overwrite) existing source file

Default Language for Code Generation: Java

Wrap long comment lines at: 80 chars(-1 = no wrap)

Auto Layout Diagram on Import: On New Diagram

[✓] Prompt when synchronizing (reversing)

[✓] Remove hard breaks from comments on import

[✓] Auto generate role names when creating code

[ ] Do not generate members where association direction is 'Unspecified'

[ ] Create dependencies for operation returns and parameter types

Comments: [✓] Generate [✓] Reverse

Remove prefixes when generating Get/Set properties (separate entries with ';'): m_s;m_n;m_b;m_;

[ ] Treat as suffixes

[ ] Capitalized Attribute Name for Properties [✓] Use 'Is' for Boolean property Get()

Code page for source:

6. Set any custom options (for this Class alone), then click on the **Close** button to return to the Generate Code dialog.
7. In the **Import(s) / Header(s)** fields, type any import statements, *#includes* or other header information. (Note that in the case of Visual Basic this information is ignored; in the case of Java the two import text boxes are merged; and in the case of C++ the first import text area is placed in the header file and the second in the body (.cpp) file.)
8. Click on the **Generate** button to create the source code.
9. When complete, click on the **View** button to see what has been generated. Note that you should set up your default viewer/editor for each language type first (see the *Source Code Viewer* topic in *Using Enterprise Architect - UML Modeling Tool)*. You can also set up the default editor on the DDL page of the Options dialog (**Tools | Options | Source Code Engineering | Code Editors**).

1.2.2 Generate a Group of Classes

In addition to being able to generate code for an individual Class, you can also select a group of Classes for batch code generation. When you do this, you accept all the default code generation options for each Class in the set.

To Generate Multiple Classes

1. Select a group of Classes and/or interfaces in a diagram.
2. Right-click on an element in the group to display the context menu.
3. Select the **Code Generation | Generate Selected elements** menu option. The Save As dialog displays, on which you specify the file path and name for each code file. Enter this information and click on the **Save** button.
4. The Batch Generation dialog displays, showing the status of the process as it executes (the process might be too fast to see this dialog).

Note:

If any of the elements selected are not Classes or interfaces the option to generate code is not available.

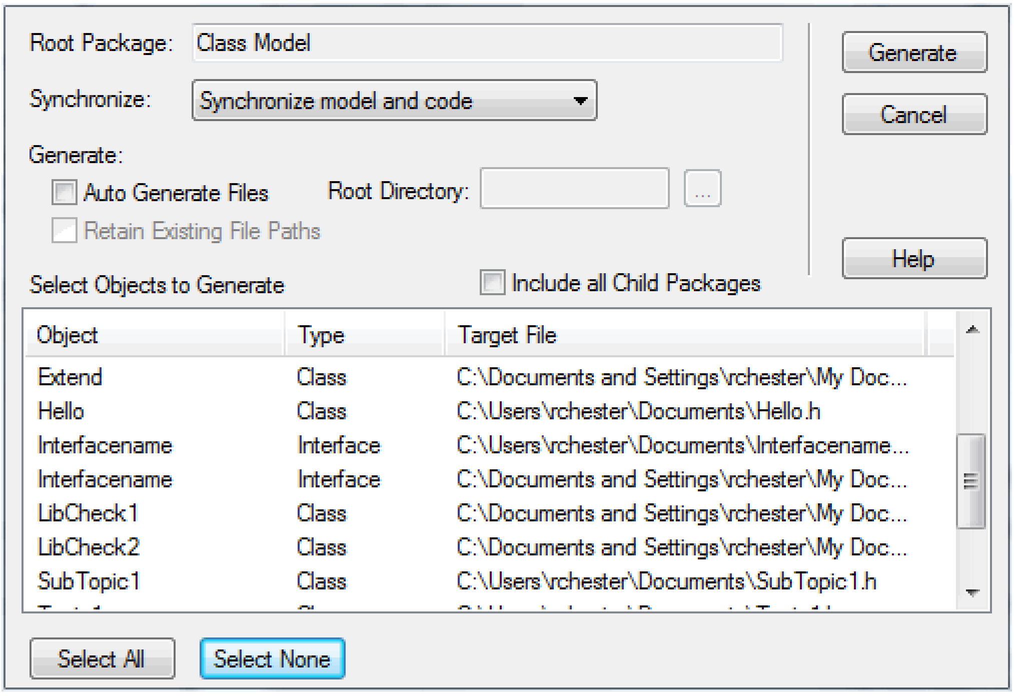
1.2.3 Generate a Package

In addition to generating source code from single Classes and groups of Classes, you can also generate code from a package. This feature provides options to recursively generate child packages and automatically generate directory structures based on the package hierarchy. This enables you to generate a whole branch of your project model in one step.

Generate a Package

To generate a package, follow the steps below:

1. In the Project Browser, right-click on the package to generate code for. The context menu displays.
2. Select the **Code Engineering | Generate Source Code** menu option. The Generate Package Source Code dialog displays.



Root Package: Class Model

Synchronize: Synchronize model and code

Generate:

Auto Generate Files

Root Directory:

Retain Existing File Paths

Select Objects to Generate

Include all Child Packages

Object	Type	Target File
Extend	Class	C:\Documents and Settings\rchester\My Doc...
Hello	Class	C:\Users\rchester\Documents\Hello.h
Interfacename	Interface	C:\Users\rchester\Documents\Interfacename...
Interfacename	Interface	C:\Documents and Settings\rchester\My Doc...
LibCheck1	Class	C:\Documents and Settings\rchester\My Doc...
LibCheck2	Class	C:\Documents and Settings\rchester\My Doc...
SubTopic1	Class	C:\Users\rchester\Documents\SubTopic1.h

3. In the **Synchronize** field, click on the drop-down arrow and select the appropriate synchronize option:
   - **Synchronize model and code**: Classes with existing files are forward synchronized with that file; Classes with no existing file are generated to the displayed target file
   - **Overwrite code**: All selected target files are overwritten (forward generated)
   - **Do not generate**: Only selected Classes that do not have an existing file are generated; all other Classes are ignored.
4. Highlight the Classes to generate. Leave unselected any to not generate. If you want to display the information in a more readable layout, you can resize the dialog and its columns.
5. To make Enterprise Architect automatically generate directories and filenames based on the package hierarchy, select the **Auto Generate Files** checkbox. This then enables the **Root Directory** field, in which you select a root directory under which the source directories are to be generated. By default, the **Auto Generate Files** feature *ignores* any file paths that are already associated with a Class. You can change this behavior by also selecting the **Retain Existing File Paths** checkbox.
6. To include all sub-packages in the output, select the **Include Child Packages** checkbox.
7. Click on the **Generate** button to start generating code.

As code generation proceeds Enterprise Architect displays progress messages. If a Class requires an output filename Enterprise Architect prompts you to enter one at the appropriate time (assuming **Auto Generate Files** is not selected). For example, if the selected Classes include partial Classes, a prompt displays to enter the filename for the second partial Class.

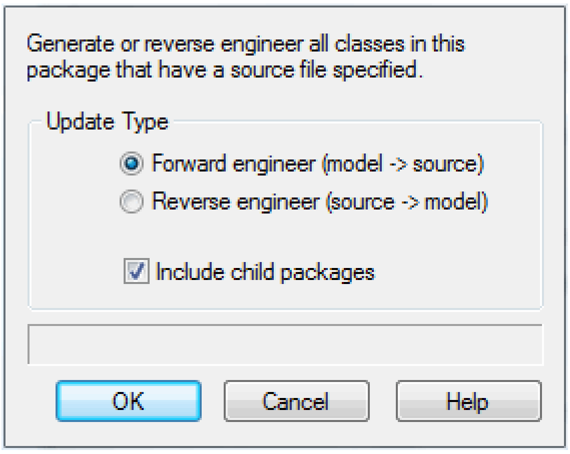
For additional information on the options on the Generate Package Source Code dialog, see the following table:

Option	Use to
Root Package	Check the name of the package to be generated.
Synchronize	Select options that specify how existing files should be generated.
Auto Generate Files	Specify whether Enterprise Architect should automatically generate file names and directories, based on the package hierarchy.
Root Directory	If **Auto Generate Files** is selected, display the path under which the generated directory structures are created.
Retain Existing File Paths	If **Auto Generate Files** is selected, specify whether to use existing file paths associated with Classes. If unselected, Enterprise Architect generates Classes to automatically determined paths, regardless of whether source files are already associated with Classes.
Include all Child Packages	Include all Classes from all sub-packages of the target package in the list. This option facilitates recursive generation of a given package and its sub-packages.
Select Objects to Generate	List all Classes that are available for generation under the target packages. Only selected (highlighted) Classes are generated. Classes are listed with their target source file.
Select All	Mark all Classes in the list as selected.
Select None	Mark all Classes in the list as unselected.
Generate	Start the generation of all selected Classes.
Cancel	Exit the Generate Package Source Code dialog. No Classes are generated.

1.2.4 Update Package Contents

Enterprise Architect enables you to synchronize a directory tree. Follow the steps below:

1. In the Project Browser, right-click on the root package of the tree to synchronize. The context menu displays.
2. Select the **Code Engineering | Synchronize Package With Code** menu option. The Synchronize Package Contents dialog displays.


Generate or reverse engineer all classes in this package that have a source file specified.

Update Type

(●) Forward engineer (model -> source)
( ) Reverse engineer (source -> model)

[✓] Include child packages

3. In the Update Type panel, select the radio button to **Forward Engineer** or **Reverse Engineer** the package Classes.
4. To include child packages in the synchronization, select the **Include child packages in generation** checkbox.
5. Click on the **OK** button to start.

Enterprise Architect uses the directory names specified when the project source was first imported/generated and updates either the model or the source code depending on the option chosen.

1.2.5 Namespaces

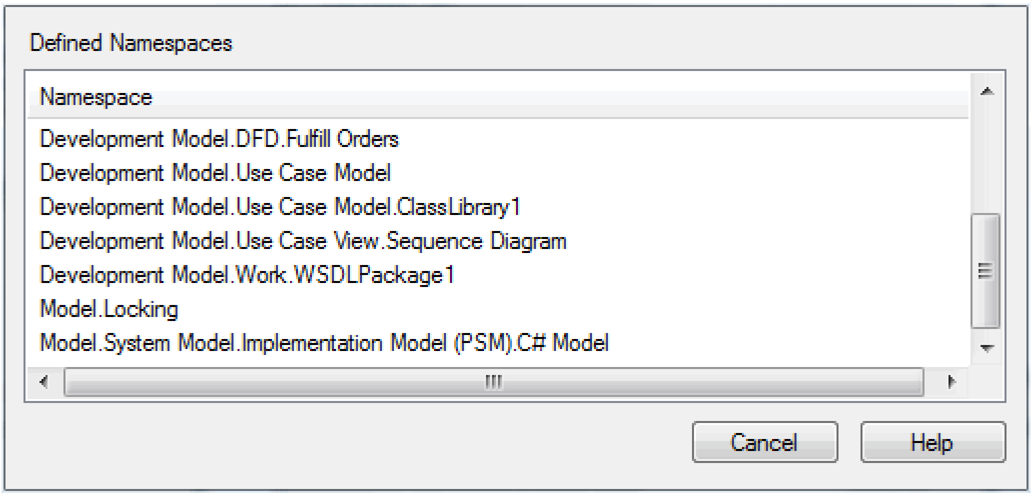
Languages such as Java support package structures or namespaces. Enterprise Architect lets you specify a package as a namespace root, which denotes where the namespace structure starts; all subordinate packages below this point are generated as namespaces to code.

To define a package as a namespace root, right-click on the package in the Project Browser and select the **Code Engineering | Set as Namespace Root** context menu option. The package icon in the Project Browser changes to include a colored corner ( ).

When you have set the namespace root, the menu option changes to **Clear Namespace Root**; click on this option to take the namespace root status off the package. (Also, see the context menu described below.)

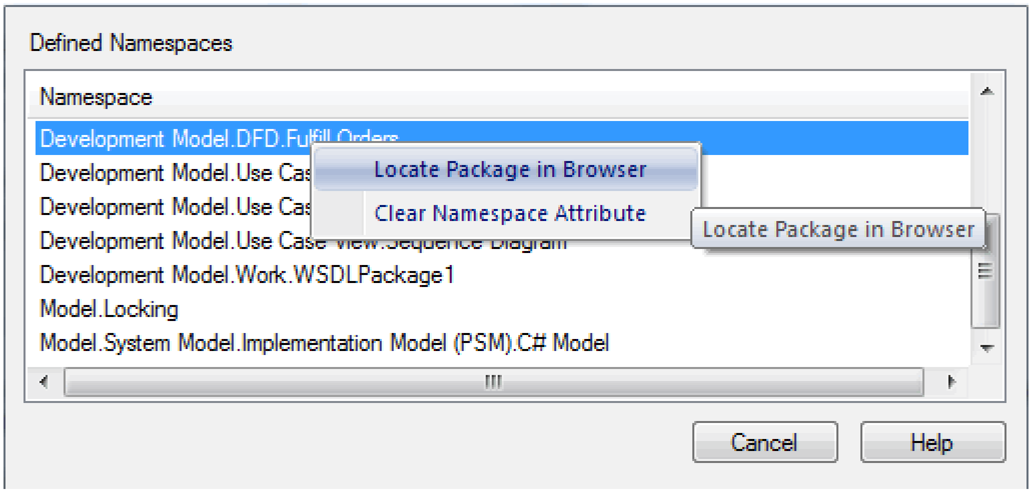
Once you have set a namespace root, Java code generated beneath this root automatically adds a package declaration at the head of the generated file indicating the current package location.

To view a list of namespaces, select the **Settings | Namespaces** menu option. The Namespaces dialog displays.


Defined Namespaces

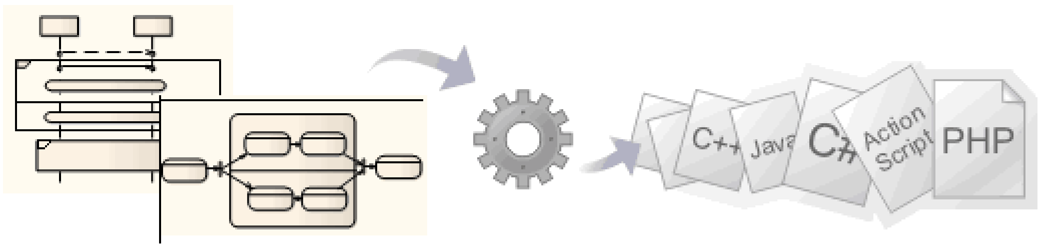
Namespace
Development Model.DFD.Fulfill Orders
Development Model.Use Case Model
Development Model.Use Case Model.ClassLibrary1
Development Model.Use Case View.Sequence Diagram
Development Model.Work.WSDLPackage1
Model.Locking
Model.System Model.Implementation Model (PSM).C# Model

If you double-click on a namespace in the list, the package is highlighted in the Project Browser. Alternatively, right-click on the namespace to display a context menu, and select the **Locate Package in Browser** menu option.


Defined Namespaces
Namespace
Development Model.DFD.Fulfill Orders
Development Model.Use Cas
Development Model.Use Cas
Development Model.Use Case View.Sequence Diagram
Development Model.Work.WSDLPackage1
Model.Locking
Model.System Model.Implementation Model (PSM).C# Model
Locate Package in Browser
Clear Namespace Attribute
Locate Package in Browser
Cancel
Help

You can also clear the selected namespace, by selecting the **Clear Namespace Attribute** option.

1.3 Code Generation From Behavioral Models



Notes:

- Software code generation from behavioral models is available in the Business and Software Engineering, Systems Engineering and Ultimate editions of Enterprise Architect.
- Hardware code generation from State Machine models is available in the Systems Engineering and Ultimate editions of Enterprise Architect.
- For C(OO), please ensure that, on the C Specifications page of the Options dialog, you have set the **Object Oriented Support** option to **True**.
- To be able to generate code from behavioral models, all behavioral constructs should be contained within a Class.

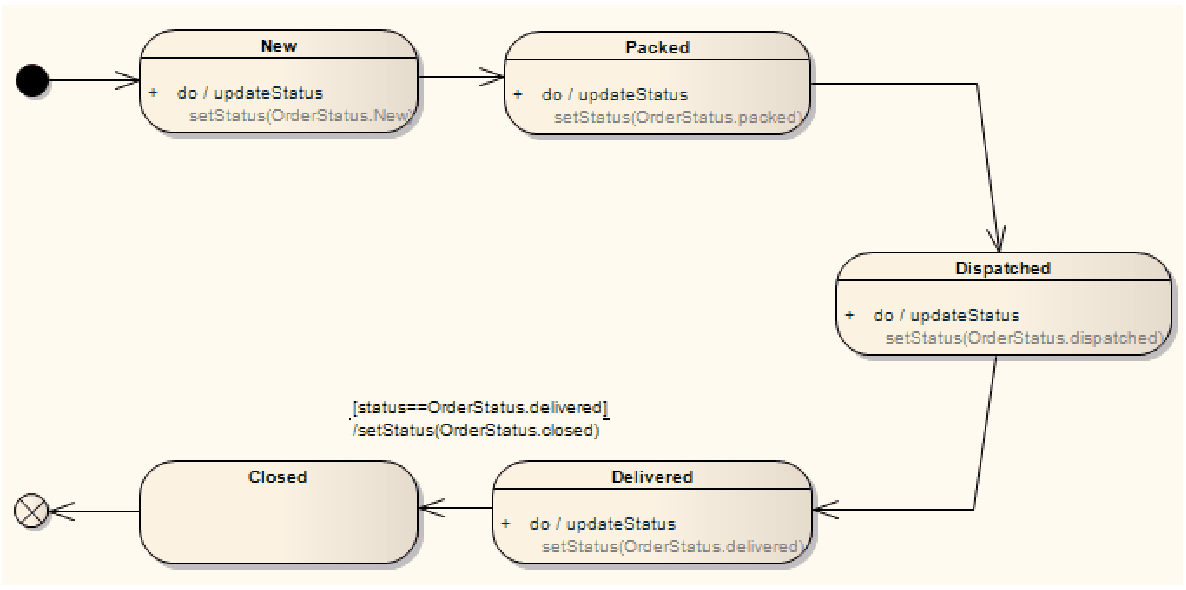
Enterprise Architect's system engineering capability facilitates code generation from each of the following UML behavioral diagrams:

- State Machine diagrams 19 (SW & HW)
- Interaction (Sequence) diagrams 26 (SW)
- Activity diagrams 27 (SW).

You can generate code in various software and hardware 23 languages, including C(OO), C++, C#, Java, VB.Net, VHDL, Verilog and SystemC.

To experiment with code generation from these diagrams, use the *EAExample* project provided with your Enterprise Architect installer.

1.3.1 SW Code Generation - State Machine Diagrams


New
+ do / updateStatus
setStatus(OrderStatus.New)
Packed
+ do / updateStatus
setStatus(OrderStatus.packed)
Dispatched
+ do / updateStatus
setStatus(OrderStatus.dispatched)
[status==OrderStatus.delivered]
/setStatus(OrderStatus.closed)
Closed
Delivered
+ do / updateStatus
setStatus(OrderStatus.delivered)

Note:

To be able to generate code from behavioral models, all behavioral constructs should be contained within a Class.

A State Machine (see the *UML Dictionary*) in a *Class* internally generates the following constructs in software languages to enable effective execution of the States' behaviors (**do**, **entry** and **exit**) and also to code the appropriate transition's effect when necessary.

Enumerations

- StateType - comprises an enumeration for each of the States contained within the State Machine
- TransitionType – comprises an enumeration for each transition that has a valid effect associated with it; for example *ProcessOrder_Delivered_to_ProcessOrder_Closed*
- CommandType – comprises an enumeration for each of the behavior types that a State can contain (**Do**, **Entry**, **Exit**).

Attributes

- currState:StateType - a variable to hold the current State's information
- nextState:StateType – a variable to hold the next State's information, set by each State's transitions accordingly
- currTransition:TransitionType – a variable to hold the current transition information; this is set if the transition has a valid effect associated with it
- transcend:Boolean - a flag used to advise if a transition is involved in transcending between different State Machines (or Submachine states)
- xx_history:StateType – a history variable for each State Machine/Submachine State, to hold information about the last State from which the transition took place.

Operations

- StatesProc - a States procedure, containing a map between a State's enumeration and its operation; it de-references the current State's information to invoke the respective State's function
- TransitionsProc - a Transitions procedure, containing a map between the Transition's enumeration and its effect; it invokes the respective effect
- <<State>> - an operation for each of the States contained within the State Machine; this renders a State's behaviors based on the input *CommandType*, and also executes its transitions
- initializeStateMachine – a function that initializes all the framework-related attributes
- runStateMachine - a function that iterates through each State, and executes their behaviors and transitions accordingly.

Click here 20 to display an example of Java code generated from the State Machine diagram above.

1.3.1.1 Java Code Generated From State Machine Diagram

```
private enum StateType : int
    {
            ProcessOrder_Delivered,
            ProcessOrder_Packed,
            ProcessOrder_Closed,
            ProcessOrder_Dispatched,
            ProcessOrder_New,
            ST_NOSTATE
    }
    private enum TransitionType : int
    {
            ProcessOrder_Delivered_to_ProcessOrder_Closed,
            TT_NOTRANSITION
    }
    private enum CommandType
    {
            Do,
            Entry,
            Exit
    }
```

```
	private StateType currState;
	private StateType nextState;
	private TransitionType currTransition;
	private boolean transcend;
	private StateType ProcessOrder_history;
	private void processOrder_Delivered(CommandType command)
	{
		switch(command)
		{
			case Do:
			{
				// Do Behaviors..
				setStatus(Delivered);
				// State's Transitions
				if((status==Delivered))
				{
					nextState = StateType.ProcessOrder_Closed;
					currTransition =
TransitionType.ProcessOrder_Delivered_to_ProcessOrder_Closed;
				}
				break;
			}
			default:
			{
				break;
			}
		}
	}

	private void processOrder_Packed(CommandType command)
	{
		switch(command)
		{
			case Do:
			{
				// Do Behaviors..
				setStatus(Packed);
				// State's Transitions
				nextState = StateType.ProcessOrder_Dispatched;
				break;
			}
			default:
			{
				break;
			}
		}
	}

	private void processOrder_Closed(CommandType command)
	{
		switch(command)
		{
			case Do:
			{
				// Do Behaviors..
				// State's Transitions
				break;
			}
			default:
			{
				break;
			}
		}
	}

	private void processOrder_Dispatched(CommandType command)
	{
		switch(command)
		{
			case Do:
			{
				// Do Behaviors..
				setStatus(Dispatched);
				// State's Transitions
				nextState = StateType.ProcessOrder_Delivered;
```

```
                        break;
                }
                default:
                {
                        break;
                }
        }
}

private void processOrder_New(CommandType command)
{
        switch(command)
        {
                case Do:
                {
                        // Do Behaviors..
                        setStatus(new);
                        // State's Transitions
                        nextState = StateType.ProcessOrder_Packed;
                        break;
                }
                default:
                {
                        break;
                }
        }
}
private void StatesProc(StateType currState, CommandType command)
{
        switch(currState)
        {
                case ProcessOrder_Delivered:
        {
                processOrder_Delivered(command);
                break;
        }

        case ProcessOrder_Packed:
        {
                processOrder_Packed(command);
                break;
        }

        case ProcessOrder_Closed:
        {
                processOrder_Closed(command);
                break;
        }

        case ProcessOrder_Dispatched:
        {
                processOrder_Dispatched(command);
                break;
        }

        case ProcessOrder_New:
        {
                processOrder_New(command);
                break;
        }
                default:
                        break;
        }
}
private void TransitionsProc(TransitionType transition)
{
        switch(transition)
        {
                case ProcessOrder_Delivered_to_ProcessOrder_Closed:
                {
                        setStatus(closed);
                        break;
                }
                default:
                        break;
```

```
            }
    }
    private void initalizeStateMachine()
    {
            currState = StateType.ProcessOrder_New;
            nextState = StateType.ST_NOSTATE;
            currTransition = TransitionType.TT_NOTRANSITION;
    }


    private void runStateMachine()
    {
            while(true)
            {
                    if ( currState == StateType.ST_NOSTATE )
                    {
                            break ;
                    }

                    currTransition = TransitionType.TT_NOTRANSITION;
                    StatesProc(currState, CommandType.Do);
    // then check if there is any valid transition assigned after the do behavior
                    if ( nextState == StateType.ST_NOSTATE)
                    {
                            break;
                    }

                    if ( currTransition != TransitionType.TT_NOTRANSITION )
                    {
                            TransitionsProc( currTransition );
                    }
                    if ( currState != nextState)
                    {
                            StatesProc(currState, CommandType.Exit);
                            StatesProc(nextState, CommandType.Entry);
                            currState = nextState ;
                    }
            }
    }
```

1.3.2 State Machine Modeling For HDLs

Note:

To be able to generate code from behavioral models, all behavioral constructs should be contained within a Class.

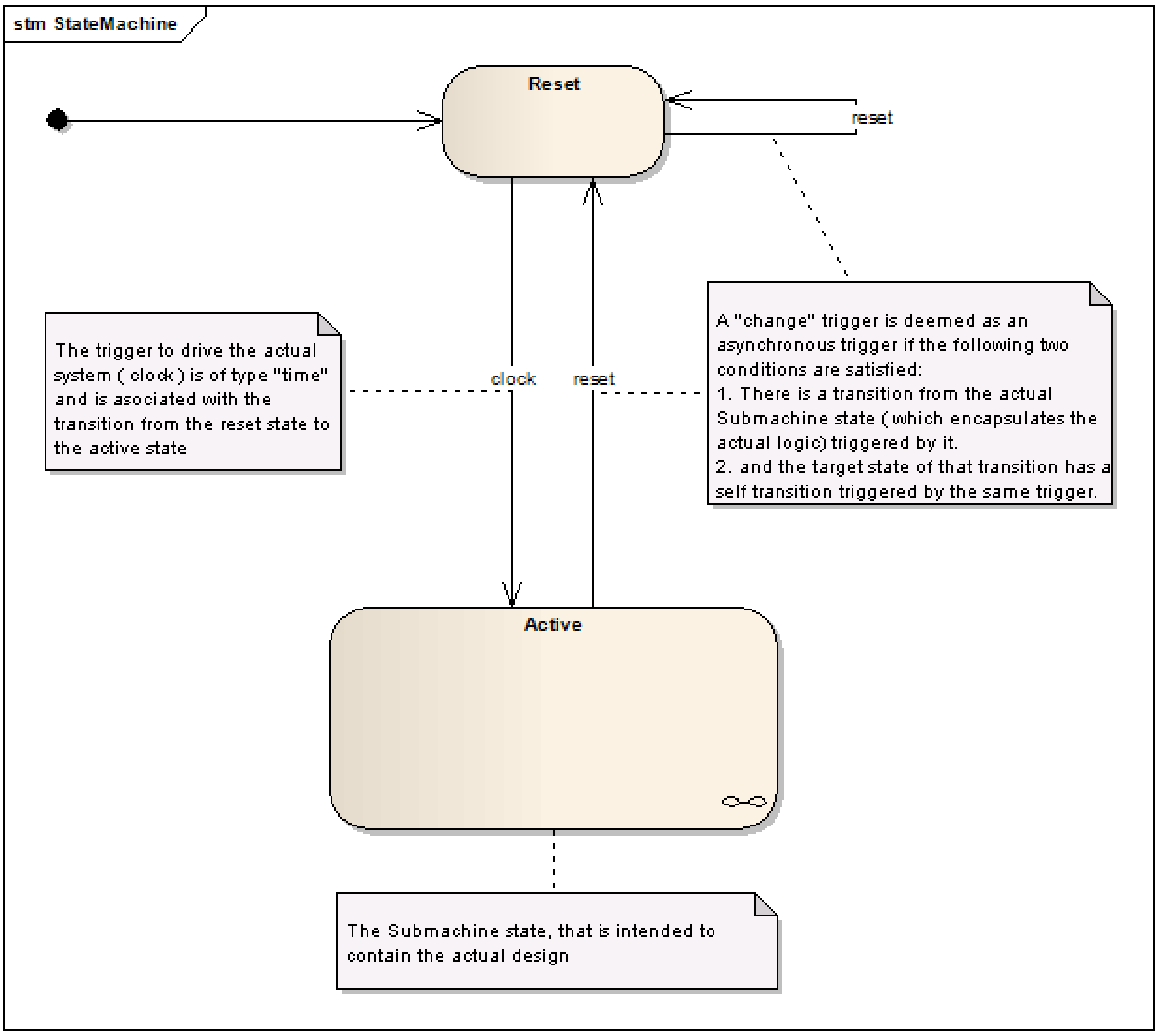
For efficient code generation from State Machine models into Hardware Description Languages (HDL) such as VHDL, Verilog and Systems C, apply the design practices outlined in this topic.

In an HDL State Machine model, the following are expected:

- Designate Driving Triggers
- Establish Port–Trigger Mapping
- Active State Logic

Designate Driving Triggers

The top level State Machine diagram should be used to model the different modes of a hardware component, and the associated triggers that drive them, as shown in the following diagram.


stm StateMachine
Reset
reset
clock
reset
Active
The trigger to drive the actual system (clock) is of type "time" and is asociated with the transition from the reset state to the active state
A "change" trigger is deemed as an asynchronous trigger if the following two conditions are satisfied:
1. There is a transition from the actual Submachine state (which encapsulates the actual logic) triggered by it.
2. and the target state of that transition has a self transition triggered by the same trigger.
The Submachine state, that is intended to contain the actual design

Asynchronous Triggers

Asynchronous triggers should be modeled according to the following pattern:

1. The trigger should be of type *Change* (specification: **true** / **false**)
2. The active state (Submachine State) should have a transition trigger by it.
3. The target state of the triggered transition should have a self transition with the same trigger

Clock

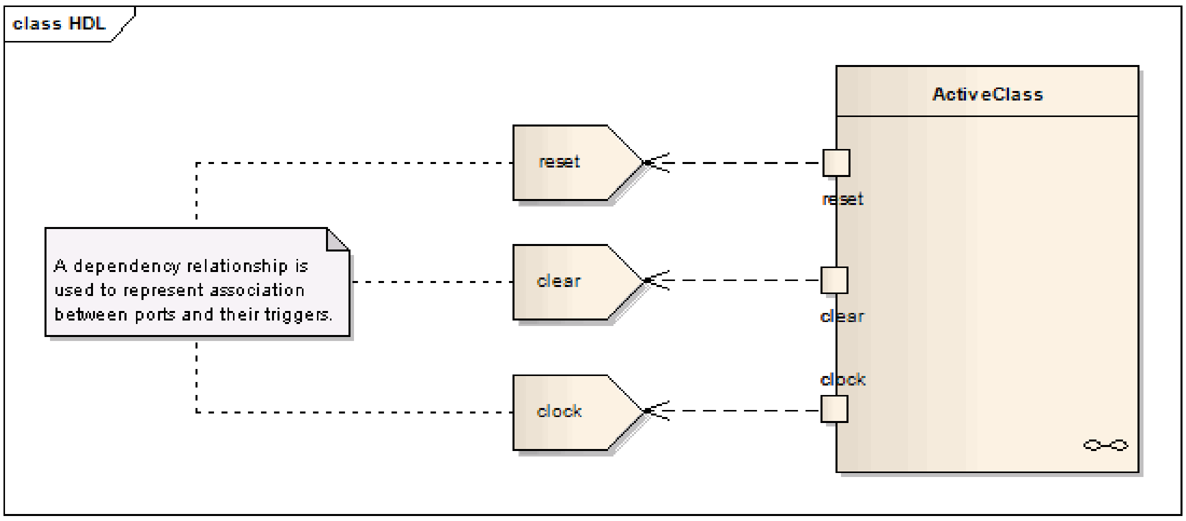
A trigger of type *time*, which triggers the transitions to the active state (Submachine State) is deemed as the *Clock*. The specification of this trigger should be specific to the target language.

Clock Trigger Specifications

Trigger Type	Language	Specification	
		Positive Edge Triggered	Negative Edge Triggered
Time	VHDL	rising_edge	falling_edge
	Verilog	posedge	negedge
	SystemC	positive	negative

Establish Port – Trigger Mapping

After successfully modeling the different operating modes of the component, and the triggers associated with them, you must associate the triggers with the component's ports as shown in the following diagram.


class HDL
ActiveClass
reset
clear
clock
reset
clear
clock
A dependency relationship is used to represent association between ports and their triggers.

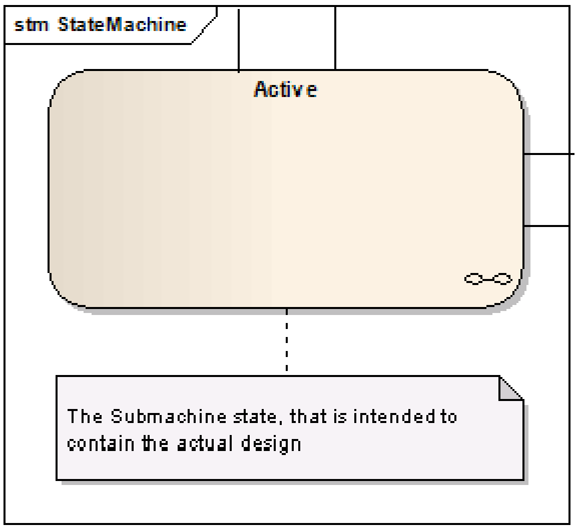
A Dependency relationship from the Port to the associated trigger should be used to signify their association.

See Also:

- State Diagram } See the *UML Dictionary*
- Transition }
- SW Code Generation - State Machine Diagrams 19.

Active State Logic

The first two aspects, above, put in place the preliminaries required for efficient interpretation of the hardware components. The actual State Machine logic is now modeled within the *Active* (Submachine) state.


stm StateMachine
Active
The Submachine state, that is intended to contain the actual design

Note:

The current code generation engine supports only one *clock trigger* for a component.

1.3.3 Code Generation - Interaction Diagrams

Note:

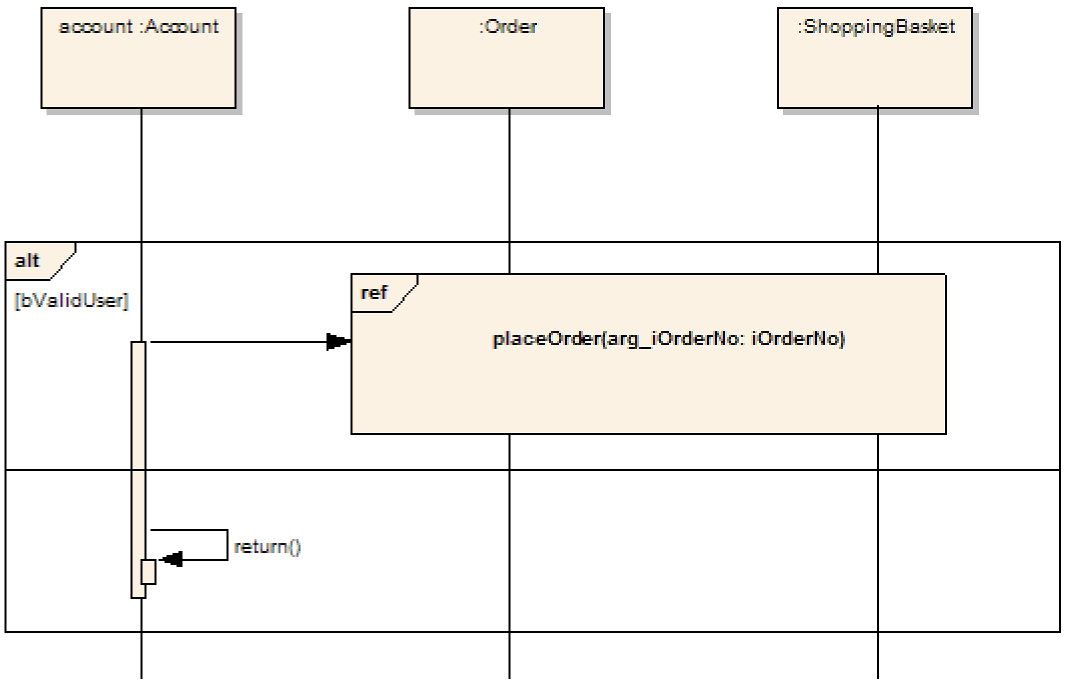
To be able to generate code from behavioral models, all behavioral constructs should be contained within a Class.

For an Interaction (Sequence) diagram, the behavioral code generation engine expects the Sequence diagram and all its associated messages and interaction fragments to be encapsulated within an Interaction element (see the *UML Dictionary*).

During code generation from Interaction (Sequence) diagrams (see the *UML Dictionary*) in a Class, Enterprise Architect applies its system engineering graph optimizer to transform the Class constructs into programmatic paradigms. Messages and Fragments are identified as one of the several action types based on their functionality, and Enterprise Architect uses the *EASL code generation templates* to render their behavior accordingly. For example:

- A Message that invokes an operation is identified as an *Action Call* and is rendered accordingly
- Combined Fragments are identified by their types and conditions; for instance, an *Alt* fragment is identified as an *Action If*, and a *loop* fragment is identified as an *Action Loop*.

For more information on the EASL code generation macros and templates Enterprise Architect uses to generate code from behavioral models see the *EASL Code Generation Macros* topic in the *Code Template Framework in SDK* section of *SDK for Enterprise Architect*.



The above diagram contains:

- A Combined Fragment (*alt*), which is identified as an *Action If* (see the *UML Dictionary*)
- An Interaction Occurrence, which is identified as an *Action Call* with all argument information associated with it (see the *UML Dictionary*), and
- A message (*Action Opaque*).

The Java code generated from this diagram resembles the following:

```
public void newTransaction()
    {
            // behavior is an Interaction
            if (bValidUser)       // Alt combined fragment
            {
                    placeOrder(101); //Interaction Occurrence
```

```
                }
                else
                {
                        return;
                }
        }
```

1.3.4 Code Generation - Activity Diagrams

Note:

To be able to generate code from behavioral models, all behavioral constructs should be contained within a Class.

Code generation from an Activity diagram (see the *UML Dictionary*) in a Class requires a validation phase, during which Enterprise Architect uses the system engineering graph optimizer to analyze the diagram and render it into various code-generatable constructs. Enterprise Architect also transforms the constructs into one of the various action types (if appropriate), similar to the Interaction diagram constructs. Enterprise Architect then uses the EASL code generation macros to generate code from these constructs.

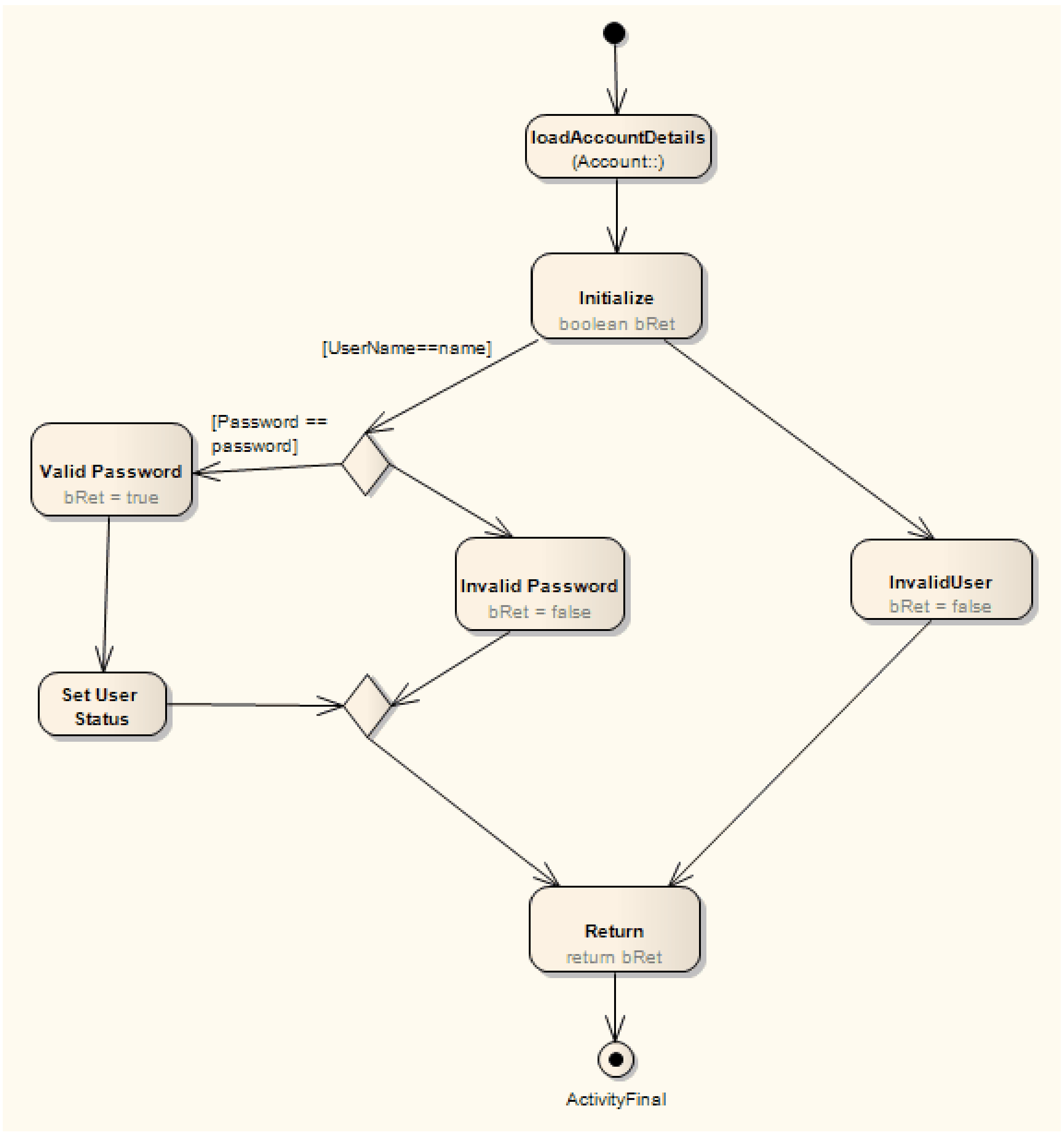
For more information on the EASL code generation macros and templates Enterprise Architect uses to generate code from behavioral models see the *EASL Code Generation Macros* topic in the *Code Template Framework in SDK* section of *SDK for Enterprise Architect*.

To provide a comprehensive analysis of these features several diagrams from the *EAExample* project are shown as examples.

Conditional Statements

To model a conditional statement, you use Decision/Merge nodes. Alternatively, you can imply Decisions/Merges internally. The graph optimizer expects an associated Merge node for each Decision node, to facilitate efficient tracking of various branches and analysis of the code constructs within them.

The following diagram is interpreted as a nested IF statement.



The Java code that might be generated from this diagram is as follows:

```
public boolean doValidateUser(String Password,String UserName)
{
        loadAccountDetails();
        boolean bRet;
        if (Username==name)
        {
                if (Password == password)
                {
                        bRet = true;
                        bValidUser = true;
                }
                else
                {
                        bRet = false;
                }

        }
        else
```

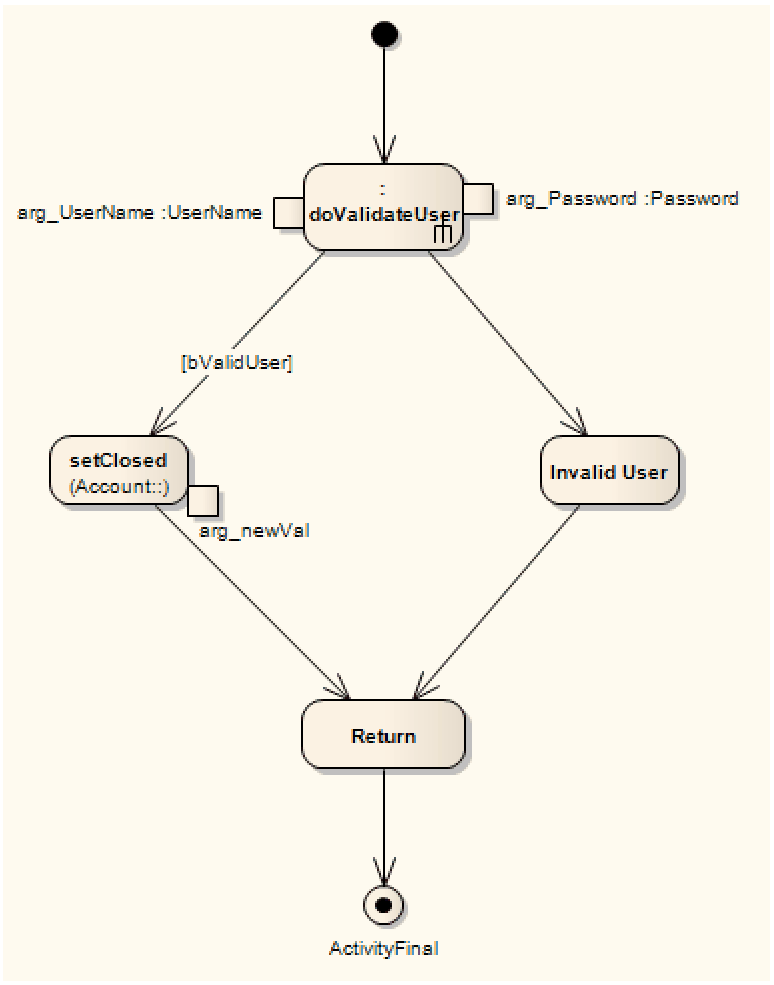
```
                {
                        bRet = false;
                }

                return bRet;
        }
```

Invocation Actions (Call Operation Action, Call Behavior Action)

Call Actions are handled more efficiently. Each action has arguments relating to the parameters of the associated behavior (use the **Synchronize** button of the Arguments dialog to synchronize arguments and parameters).

The following diagram demonstrates the use of a *Call Behavior Action* and a *Call Operation Action* interspersed with a conditional statement.



The generated Java code might appear as follows:

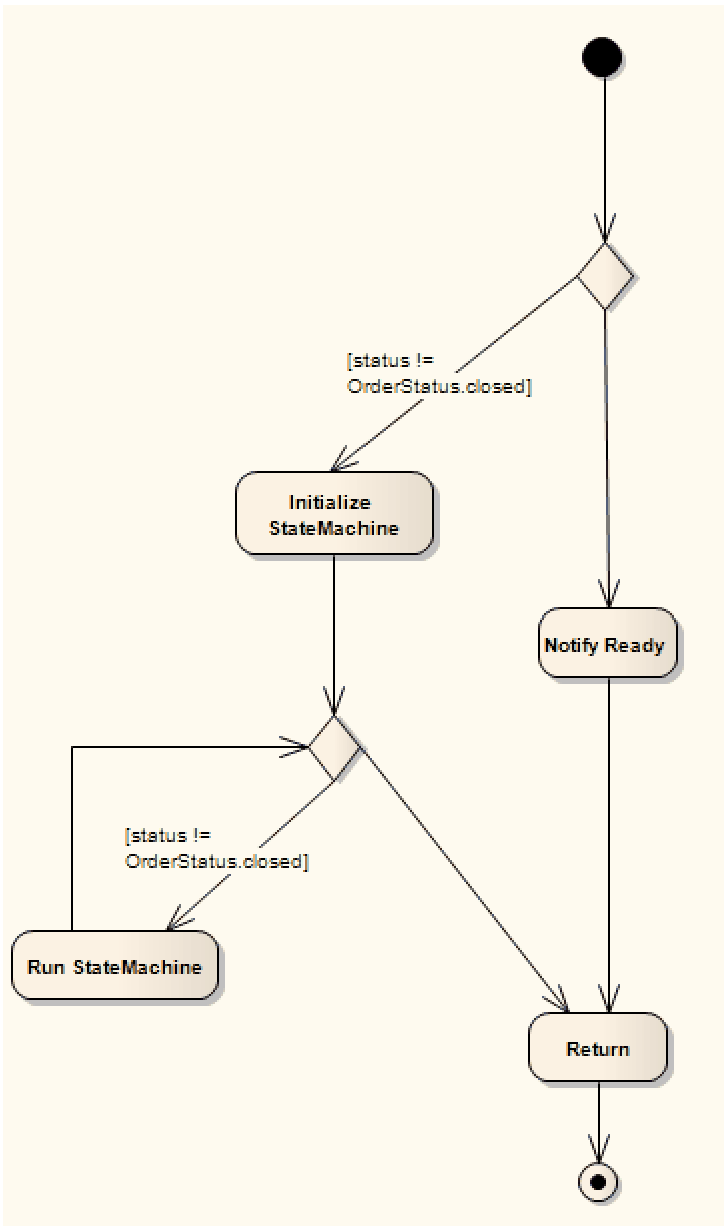
```
        public void doMarkAccountClosed()
        {
                doValidateUser(password,name);
                if (bValiduser)
                {
                        setClosed(true);

                }
                else
                {
                        System.out.println("Invalid user");
                }
                return;
        }
```

Loops

Enterprise Architect's system engineering graph optimizer is also capable of analyzing and identifying loops. An identified loop is internally rendered as an *Action Loop*, which is translated by the EASL code generation macros to generate the required code.

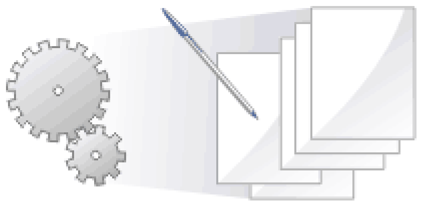
The following diagram demonstrates how a loop can be modeled.


[status != OrderStatus.closed]
Initialize StateMachine
Notify Ready
[status != OrderStatus.closed]
Run StateMachine
Return

The generated Java code might appear as follows:

```
public void doCheckForOutstandingOrders()
{
        if (status != closed)
        {
                initializeStateMachine();
                while (status != closed)
                {
                        runStateMachine();
                        }
                }
                else
                {
                        //No Outstanding orders;
                }
                return;
        }
```

1.4 Code Engineering Settings



You can set the default code options such as the editors for each of the programming languages available for Enterprise Architect and special options for how source code is generated.

See Also

- General Options 32
- Local Paths 38
- Local Path Dialog 39
- Language Macros 39
- Setting Collection Classes 41

1.4.1 Source Code Engineering

The following topics describe general options that apply to all languages when generating code from Enterprise Architect. These options are all available under the Source Code Engineering section of the Options dialog (select the **Tools | Options | Source Code Engineering** menu option).

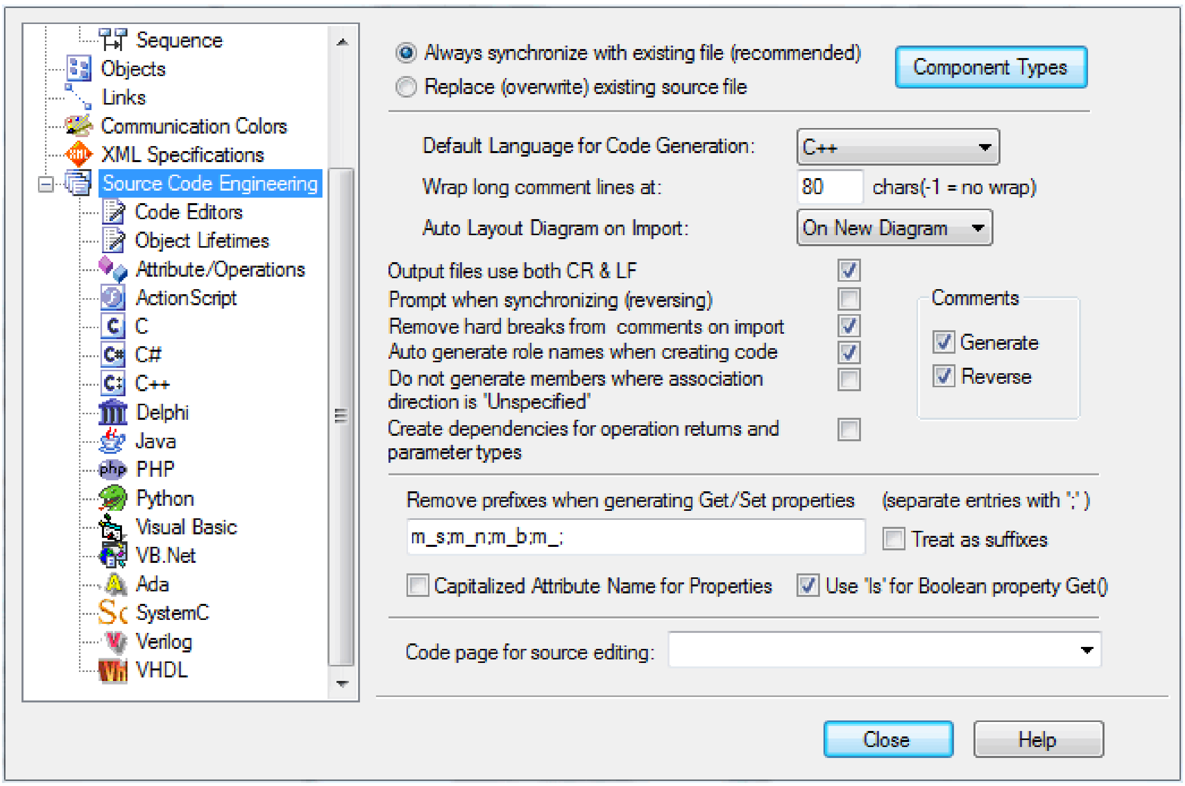
- Source Code Options 32
- Options - Code Editors 34
- Options - Object Lifetimes 35
- Options - Attribute/Operations 36
- Synchronize Model and Code 10
- Code Page for Source Editing 37

1.4.1.1 Source Code Options

When you generate code for a particular language, you can set certain options. These include:

- Create a default constructor
- Create a destructor
- Generate copy constructor
- Select default language
- Generate methods for implemented interfaces
- Set the unicode options for code generation.

These options are accessed the Source Code Engineering page of the Options dialog (select the **Tools | Options | Source Code Engineering** menu option).



(● Always synchronize with existing file (recommended)) ○ Replace (overwrite) existing source file

Default Language for Code Generation: C++

Wrap long comment lines at: 80 chars(-1 = no wrap)

Auto Layout Diagram on Import: On New Diagram

Output files use both CR & LF	✓
Prompt when synchronizing (reversing)	
Remove hard breaks from comments on import	✓
Auto generate role names when creating code	✓
Do not generate members where association direction is 'Unspecified'	
Create dependencies for operation returns and parameter types	

Comments: ✓ Generate ✓ Reverse

Remove prefixes when generating Get/Set properties (separate entries with ';')

m_s;m_n;m_b;m_;

☐ Treat as suffixes

☐ Capitalized Attribute Name for Properties ✓ Use 'Is' for Boolean property Get()

Code page for source editing:

Most of the settings are self-explanatory. The **Remove prefixes when generating Get/Set properties** field enables you to specify prefixes used in your variable naming conventions, if those prefixes should be removed in the variables' corresponding get/set functions.

Click on the **Component Types** button to configure what elements 33 you would like to be created for files of any extension found while importing a source code directory.

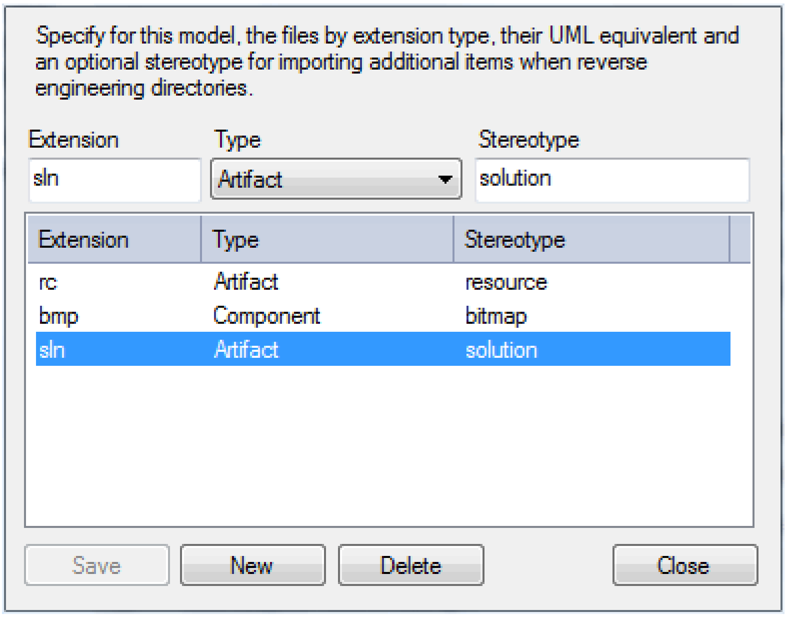
Note:

It is worthwhile to configure these settings, as they serve as the defaults for all Classes in the model. You can override these on a per-Class basis using the custom settings (from the Code Generation dialog).

1.4.1.1.1 *Import Component Types*

The Import Component Types dialog enables you to configure what elements you would like to be created for files of any extension found while importing a source code directory.

To access the Import Component Types dialog select the **Tools | Options | Source Code Engineering** menu option to display the Source Code Engineering page of the Options dialog, and click on the **Component Types** button.



Specify for this model, the files by extension type, their UML equivalent and an optional stereotype for importing additional items when reverse engineering directories.

Extension: sln

Type: Artifact

Stereotype: solution

Extension	Type	Stereotype
rc	Artifact	resource
bmp	Component	bitmap
sln	Artifact	solution

For each extension you can specify:

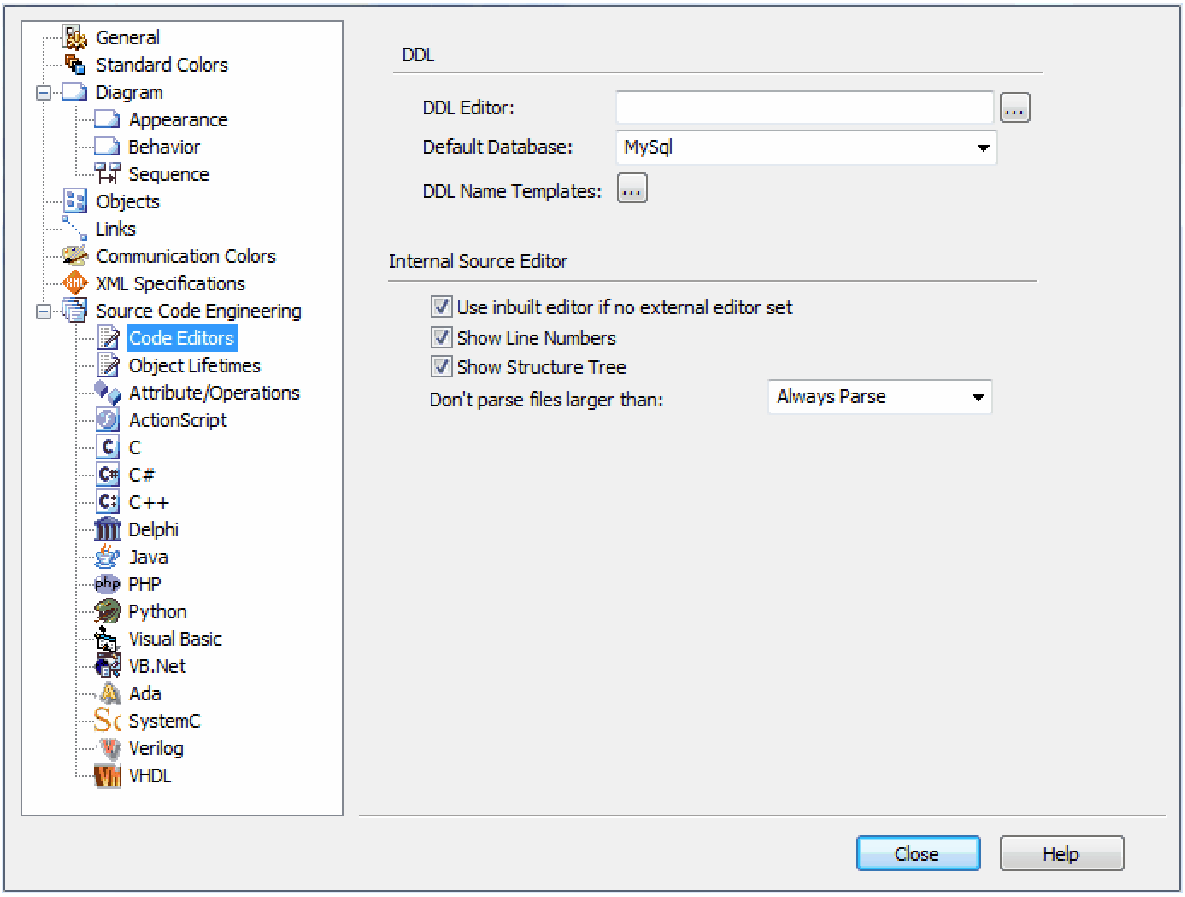
- The element type to be created
- The stereotype to apply to these objects.

Note:

You can transport these import component types between models, using the **Export Reference Data** and **Import Reference Data** options on the **Tools** menu. See the *Reference Data* topic in *UML Model Management.*

1.4.1.2 Options - Code Editors

You access the source code editor options via the DDL page of the Options dialog (select the **Tools | Options | Source Code Engineering | Code Editors** menu option). They enable you to configure options for Enterprise Architect's internal editor, as well as the default editor for DDL scripts. You can configure external editors for code languages on each language options page.



DDL

DDL Editor:

Default Database: MySql

DDL Name Templates:

Internal Source Editor

☑ Use inbuilt editor if no external editor set

☑ Show Line Numbers

☑ Show Structure Tree

Don't parse files larger than: Always Parse

The options for the inbuilt editor are:

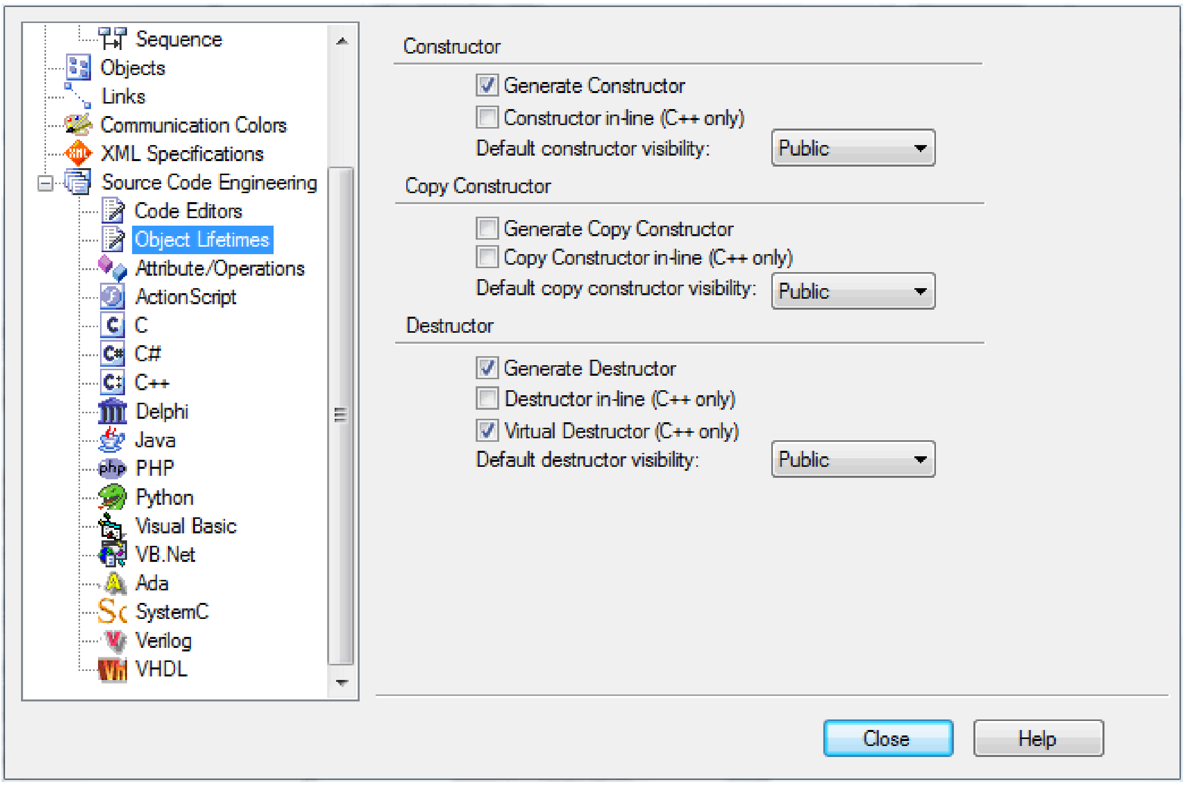
Option	Use to
Use inbuilt editor if no external editor set	Specify the editor for code in a language if no external editor is defined for that language.
Show Line Numbers	Show line numbers in the editor.
Show Structure Tree	Show a tree with the results of parsing the open file (requires that the file is parsed successfully).
Don't parse files larger than	Specify an upper limit on file size for parsing. Used to prevent performance decrease due to parsing very large files.

1.4.1.3 Options - Object Lifetimes

This set of options enables you to configure:

- Constructor details when generating code
- Whether to create a copy constructor
- Destructor details.

These options are accessed via the Constructor page of the Options dialog (select the **Tools | Options | Source Code Engineering | Object Lifetimes** menu option).



Constructor

- [✓] Generate Constructor
- [ ] Constructor in-line (C++ only)

Default constructor visibility: Public

Copy Constructor

- [ ] Generate Copy Constructor
- [ ] Copy Constructor in-line (C++ only)

Default copy constructor visibility: Public

Destructor

- [✓] Generate Destructor
- [ ] Destructor in-line (C++ only)
- [✓] Virtual Destructor (C++ only)

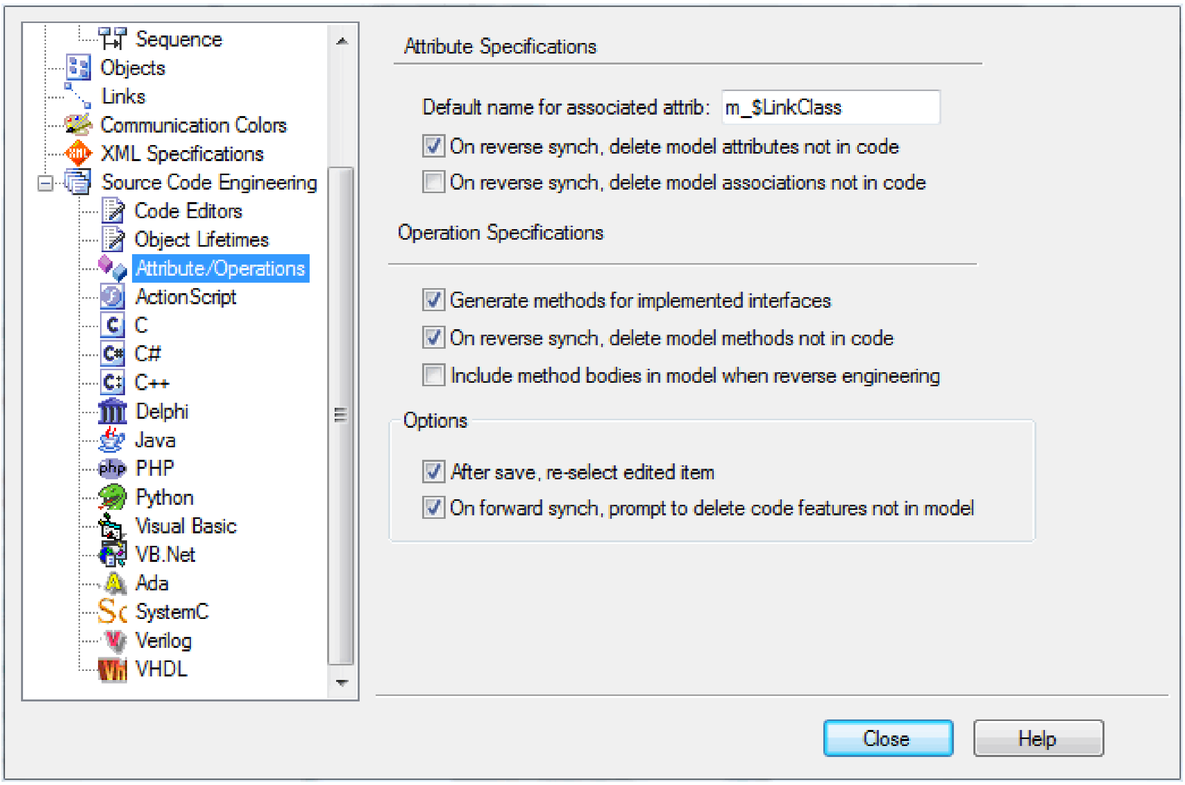
Default destructor visibility: Public

1.4.1.4 Options - Attribute/Operations

This set of options enables you to:

- Configure the default name generated from imported attributes
- Generate methods for implemented interfaces
- Delete model attributes not included in the code during reverse synchronization
- Delete model methods not included in the code during reverse synchronization
- Delete code from features contained in the model during forward synchronization
- Delete model associations and aggregations that correspond to attributes not included in the code during reverse synchronization
- Define whether or not the bodies of methods are included and saved in the Enterprise Architect model when reverse engineering
- Create attributes in quick succession, clearing the dialog when you click on **Save** so that you can enter another attribute name.

These options are accessed via the Attribute Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | Attribute/Operations** menu option).



Attribute Specifications

Default name for associated attrib: m_$LinkClass

✓ On reverse synch, delete model attributes not in code

☐ On reverse synch, delete model associations not in code

Operation Specifications

✓ Generate methods for implemented interfaces

✓ On reverse synch, delete model methods not in code

☐ Include method bodies in model when reverse engineering

Options

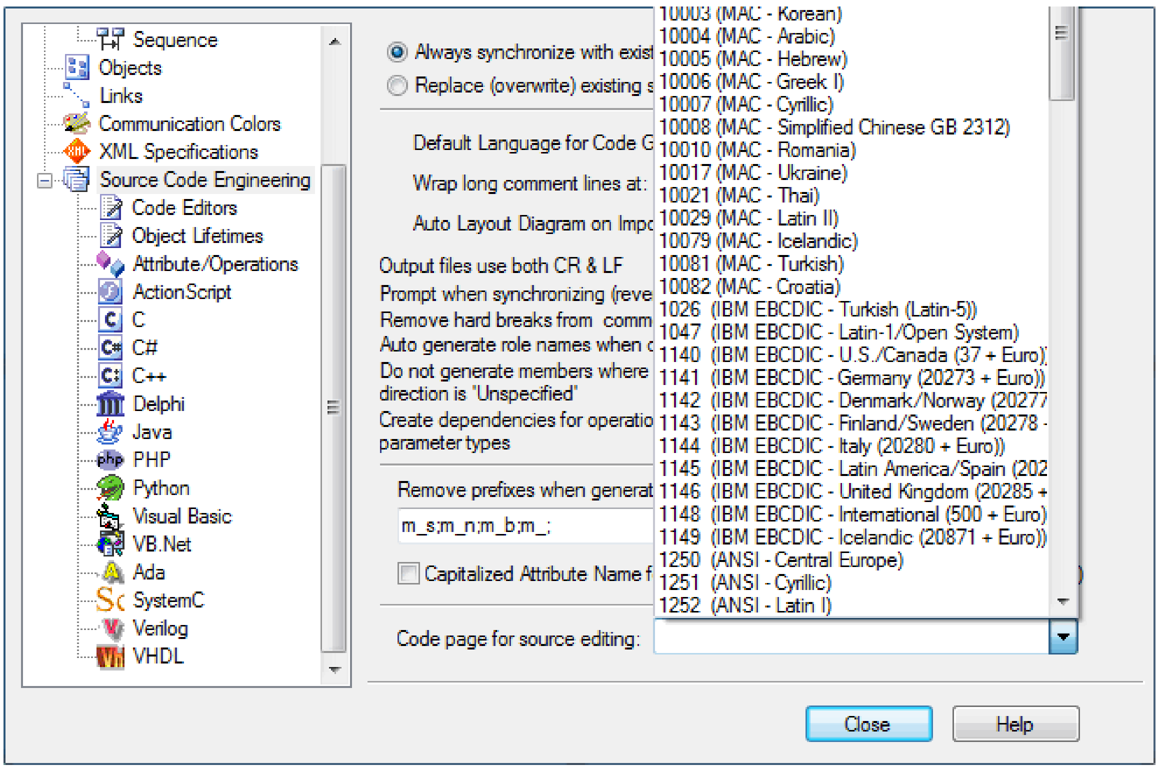
✓ After save, re-select edited item

✓ On forward synch, prompt to delete code features not in model

1.4.1.5 Code Page for Source Editing

Enterprise Architect enables you to define the Unicode character set for code generation. To set the Unicode character set follow the steps below:

1. Select the **Tools | Options | Source Code Engineering** menu option. The Source Code Engineering page of the Options dialog displays.



Always synchronize with exist
Replace (overwrite) existing s
Default Language for Code G
Wrap long comment lines at:
Auto Layout Diagram on Impo
Output files use both CR & LF
Prompt when synchronizing (reve
Remove hard breaks from comm
Auto generate role names when c
Do not generate members where direction is 'Unspecified'
Create dependencies for operatio parameter types
Remove prefixes when generat
m_s;m_n;m_b;m_;
Capitalized Attribute Name f
Code page for source editing:

10003 (MAC - Korean)
10004 (MAC - Arabic)
10005 (MAC - Hebrew)
10006 (MAC - Greek I)
10007 (MAC - Cyrillic)
10008 (MAC - Simplified Chinese GB 2312)
10010 (MAC - Romania)
10017 (MAC - Ukraine)
10021 (MAC - Thai)
10029 (MAC - Latin II)
10079 (MAC - Icelandic)
10081 (MAC - Turkish)
10082 (MAC - Croatia)
1026 (IBM EBCDIC - Turkish (Latin-5))
1047 (IBM EBCDIC - Latin-1/Open System)
1140 (IBM EBCDIC - U.S./Canada (37 + Euro)
1141 (IBM EBCDIC - Germany (20273 + Euro))
1142 (IBM EBCDIC - Denmark/Norway (20277
1143 (IBM EBCDIC - Finland/Sweden (20278 -
1144 (IBM EBCDIC - Italy (20280 + Euro))
1145 (IBM EBCDIC - Latin America/Spain (202
1146 (IBM EBCDIC - United Kingdom (20285 +
1148 (IBM EBCDIC - International (500 + Euro)
1149 (IBM EBCDIC - Icelandic (20871 + Euro))
1250 (ANSI - Central Europe)
1251 (ANSI - Cyrillic)
1252 (ANSI - Latin I)

2. In the **Code page for source editing** field, click on the drop-down arrow and select the appropriate Unicode character set.
3. Click on the **Close** button.

1.4.2 Local Paths

Sometimes a team of developers could be working on the same Enterprise Architect model. Each developer might store their version of the source code in their local file system, but not always at the same location as their fellow developers. To handle this scenario, Enterprise Architect enables you to define local paths for each Enterprise Architect user, using the Local Paths 39 dialog (select the **Settings | Local Paths** menu option).

As well as generating code and reverse engineering, you can use local paths in version control, developing XML schemas, and generating RTF and HTML reports.

Local paths take a bit of setting up, but if you want to work collaboratively on source and model concurrently, the effort is well worth while.

For example: developer A stores their .java files in a C:\Java\Source directory, while developer B stores theirs in D:\Source. Meanwhile, both developers want to generate and reverse engineer into the same Enterprise Architect model located on a shared (or replicated) network drive.

Developer A might define a local path of:

JAVA_SOURCE = "C:\Java\Source"

All Classes generated and stored in the Enterprise Architect project are stored as:

%JAVA_SOURCE%\<xxx.java>.

Developer B now defines a local path as:

JAVA_SOURCE ="D:\Source".

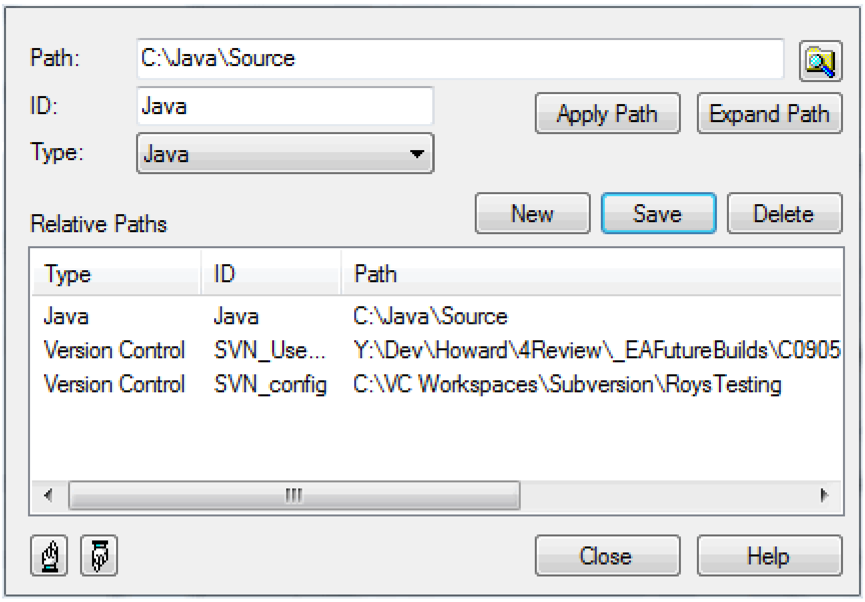
Now, Enterprise Architect stores all java files in these directories as:

%JAVA_SOURCE%\<filename>

On each developer's machine, the filename is expanded to the correct local version.

1.4.3 Local Paths Dialog

The Local Paths dialog enables you to set up local paths for a single user on a particular machine. For a description of what Local Paths are used for, see the *Local Paths* 38 topic. To open the Local Paths dialog, select the **Settings | Local Paths** option.



Path: C:\Java\Source
ID: Java
Type: Java

Type	ID	Path
Java	Java	C:\Java\Source
Version Control	SVN_Use...	Y:\Dev\Howard\4Review_EAFutureBuilds\C0905
Version Control	SVN_config	C:\VC Workspaces\Subversion\RoysTesting

The Local Paths dialog enables you to define:

- **Path -** the local directory in the file system (for example, d:\java\source)
- **ID** - the shared ID that is substituted for the Local Path (for example, JAVA_SRC)
- **Type** - the language type (for example, Java).

And also to:

- **Apply Path** - Select a path and click on this button to update any existing paths in the model (with full path names) to the shared relative path name (so d:\java\source\main.java might become %JAVA_SRC%\main.java)
- **Expand Path -** The opposite of **Apply Path**. This enables you to remove a relative path and substitute the full path name.

Using the two options you can update and change existing paths.

Note:

You can also set up a hyperlink (see the *UML Dictionary*) on a diagram to access the Local Paths dialog, to switch, update or expand your current Local Path.

1.4.4 Language Macros

When reverse engineering a language such as C++, you might find preprocessor directives scattered throughout the code. This can make code management easier, but can hamper parsing of the underlying C++ language.

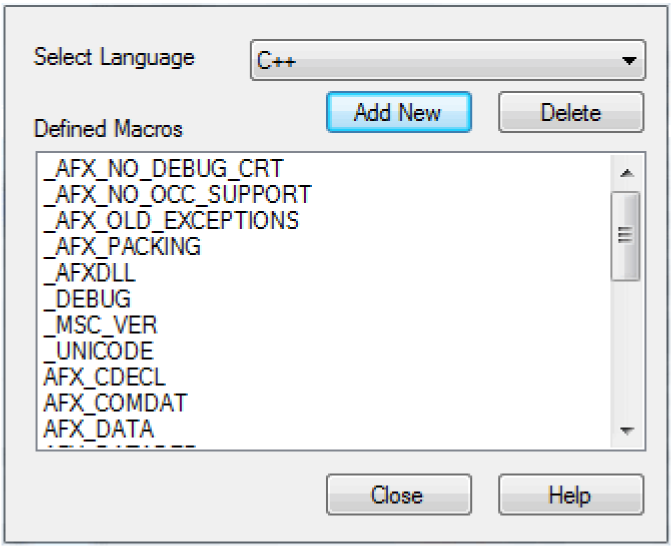
To help remedy this, you can include any number of *macro* definitions, which are ignored during the parsing phase of the reverse engineering. It is still preferable, if you have the facility, to preprocess the code using the appropriate compiler first; this way, complex macro definitions and defines are expanded out and can be readily parsed. If you don't have this facility, then this option provides a convenient substitute.

Note:

You can transport these language macro (or preprocessor macro) definitions between models, using the **Export Reference Data** and **Import Reference Data** options on the **Tools** menu. The macros are exported as a *Macro List*. See the *Reference Data* topic in *UML Model Management*.

Define a Macro

1. Select the **Settings | Preprocessor Macros** menu option. The Language Macros dialog displays.



```
_AFX_NO_DEBUG_CRT
_AFX_NO_OCC_SUPPORT
_AFX_OLD_EXCEPTIONS
_AFX_PACKING
_AFXDLL
_DEBUG
_MSC_VER
_UNICODE
AFX_CDECL
AFX_COMDAT
AFX_DATA
```

2. Click on the **Add New** button.
3. Enter details for your macro.
4. Click on the **OK** button.

Macros Embedded Within Declarations

Macros are sometimes used within the declaration of Classes and operations, as in the following examples:

```
class __declspec Foo
{
    int __declspec Bar(int p);
};
```

If *declspec* is defined as a C++ macro, as outlined above, the imported Class and operation contain a Tagged Value called *DeclMacro1* with value *__declspec*. (Subsequent macros would be defined as *DeclMacro2*, *DeclMacro3* and so on.) During forward engineering, these Tagged Values are used to regenerate the macros in code.

Define Complex Macros

It is sometimes useful to define rules for complex macros that can span multiple lines. Enterprise Architect ignores the entire code section defined by the rule. Such macros can be defined in Enterprise Architect as in the following two examples. Both types can be combined in one definition.

Block Macros

```
BEGIN_INTERFACE_PART ^ END_INTERFACE_PART
```

where the **^** symbol represents the body of the macro. This enables skipping from one macro to another.

Note:

The spaces surrounding the ^ symbol are required.

Function Macros

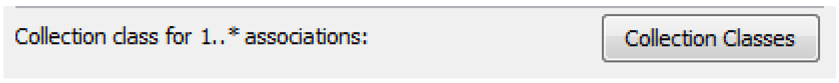
```
RTTI_EMULATION()
```

where Enterprise Architect skips over the token including everything inside the parentheses.

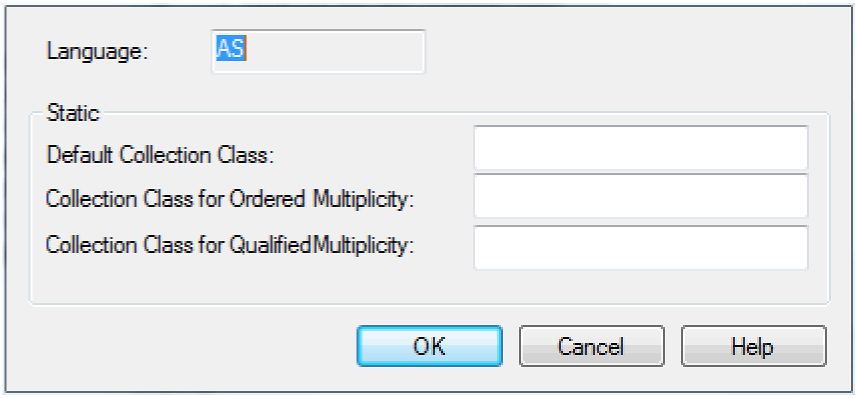
1.4.5 Set Collection Classes

Enterprise Architect enables you to define *Collection Classes* for generating code from Association connectors where the target role has a multiplicity setting greater than 1. There are two options for doing this:

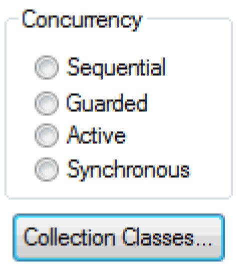
1. On the Source Code Engineering section of the Options dialog (select the **Tools | Options | Source Code Engineering** option), on each language page click on the **Collection Classes** button.

   The Collection Classes for Association Roles dialog displays.

   
   Language:
   AS
   Static
   Default Collection Class:
   Collection Class for Ordered Multiplicity:
   Collection Class for Qualified Multiplicity:
   OK
   Cancel
   Help

   On this dialog, you can define:
   - The default Collection Class for 1..* roles
   - The ordered Collection Class to use for 1..* roles
   - The qualified Collection Class to use for 1..* roles.
2. On the Detail tab of the Class Properties dialog (accessible from the right-click context menu of any Class), click on the **Collection Classes** button.

   The Collection Classes for Association Roles dialog again displays, but here you define ***for when only this Class is used***:
   - The default Collection Class for 1..* roles
   - The ordered Collection Class to use for 1..* roles
   - The qualified Collection Class to use for 1..* roles.

When Enterprise Architect generates code for a connector that has a multiplicity role >1, the Collection Class is calculated as follows:

1. If the **Qualifier** is set use the qualified collection:
   - for the Class if set
   - else use the code language qualified collection.
2. If the **Order** option is set use the ordered collection:
   - for the Class if set

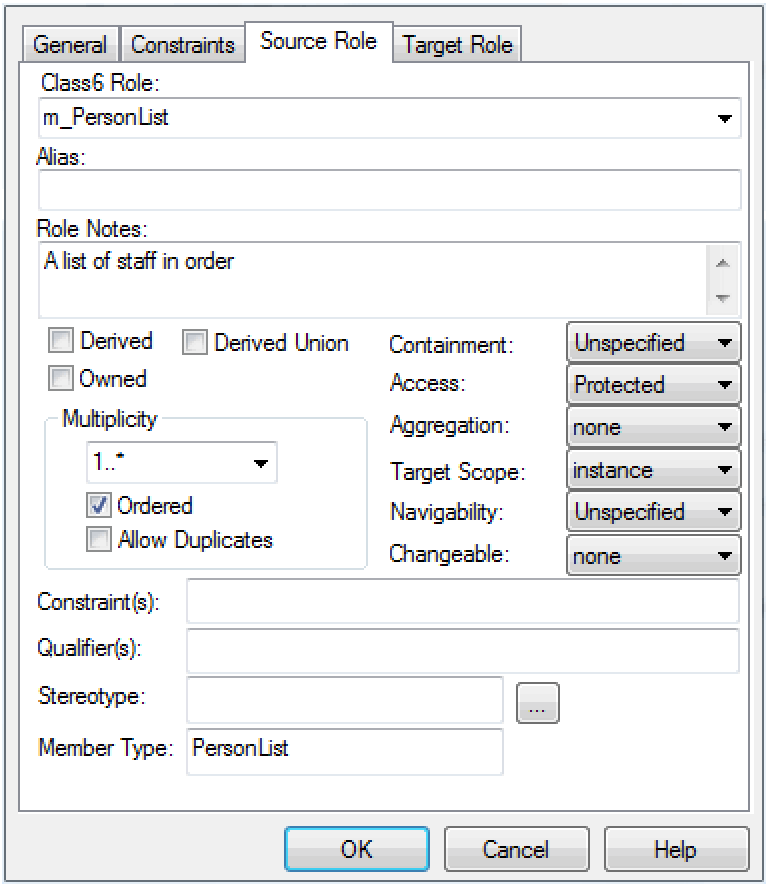
- else use the code language ordered collection.

3. Else use the default collection:
   - for the Class if set
   - else use the code language default collection.

Note:

You can include the marker #TYPE# in the collection name; Enterprise Architect replaces this with the name of the Class being collected at source generation time (for example, Vector<#TYPE#> would become Vector<foo>).

Additionally, on both the Source Role and Target Role tabs of the Association Property dialog (accessible from the right-click context menu of any Association) there is a **Member Type** field. If you set this, the value you enter overrides all the above options. The example below shows a defined *PersonList;* when code is generated, because this has a **Multiplicity** greater than 1 and the **Member Type** is defined, the variable created is of type *PersonList*.



Class6 Role: m_PersonList
Alias:
Role Notes: A list of staff in order
[ ] Derived [ ] Derived Union
[ ] Owned
Multiplicity: 1..*
[✓] Ordered
[ ] Allow Duplicates
Containment: Unspecified
Access: Protected
Aggregation: none
Target Scope: instance
Navigability: Unspecified
Changeable: none
Constraint(s):
Qualifier(s):
Stereotype:
Member Type: PersonList

1.4.6 Language Options

You can set up various options for how Enterprise Architect handles a particular language when generating code. These options are accessible on the Options dialog (select the **Tools | Options** menu option).

Under the **Source Code Engineering** option, select the required language. The following topics outline the options available for each language.

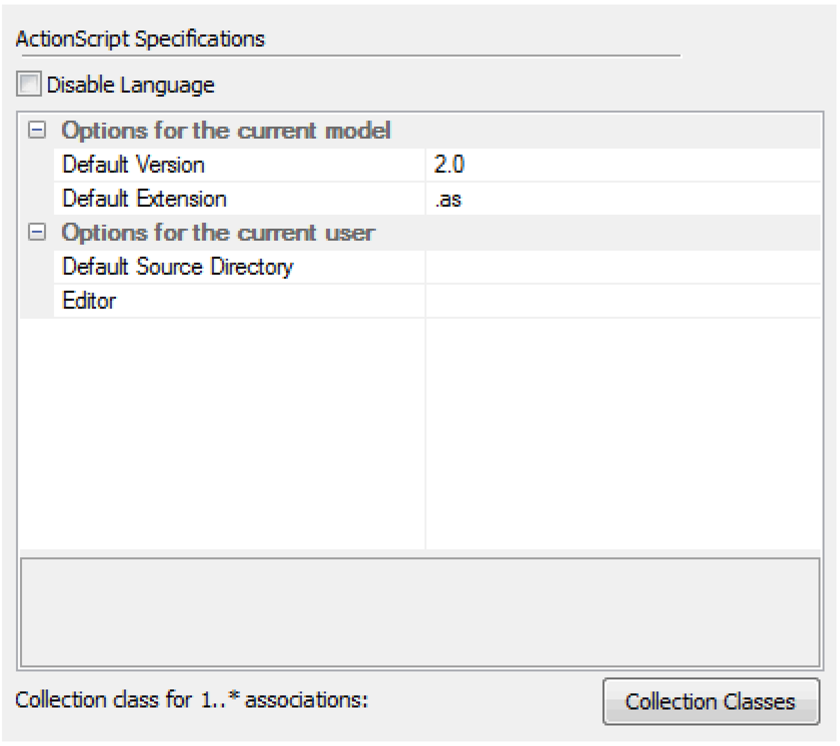
- ActionScript 43
- Ada 2005 43 (in the System Engineering and Ultimate editions of Enterprise Architect)
- ANSI C 44
- C# 45
- C++ 46

- Delphi 47
- Delphi Properties 48
- Java 51
- PHP 51
- Python 52
- SystemC 53
- VB.Net 54
- Verilog 55
- VHDL 56
- Visual Basic 57
- MDG Technology Languages 58
- Reset Options 59

1.4.6.1 ActionScript Options

Configure options for ActionScript code generation using the ActionScript Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | ActionScript** menu option). The options you can specify include the:

- Default ActionScript version to generate (AS2.0 or AS3.0)
- Default file extensions (header and source)
- Default source directory
- Editor for ActionScript code.


ActionScript Specifications

☐ Disable Language

Options for the current model	
Default Version	2.0
Default Extension	.as
Options for the current user	
Default Source Directory	
Editor	

Collection class for 1..* associations:

1.4.6.2 Ada 2005 Options

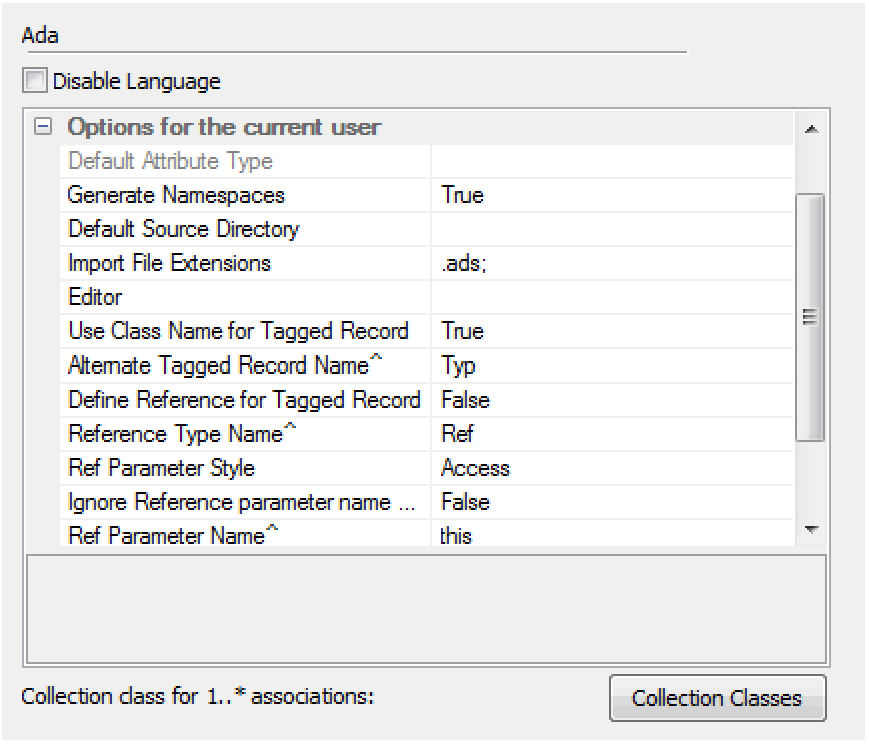
Note:

Ada 2005 support is available in the System Engineering and Ultimate editions of Enterprise Architect.

Configure options for Ada 2005 code generation using the Ada page of the Options dialog (select the **Tools |**

Options | Source Code Engineering | Ada menu option). The options you can specify include:

- Use Class Name for Tagged Record - to inform the reverse engineering process whether the name of the Tagged Record is the same as the package name
- Alternate Tagged Record Name - to advise the engine of the alternate Tagged Record name to locate
- Define Reference for Tagged Record - to specify whether the engine should create a reference type for the Tagged Record (if one is not defined)
- Reference Type Name - to supply the name of the reference type to be created (default is *Ref*)
- Ref Parameter Style - to specify the reference parameter of a Reference / Access type
- Ignore Reference Parameter Name - to tell the engine to ignore the name of the reference parameter
- Ref Parameter Name - to indicate the name of the reference parameter to locate

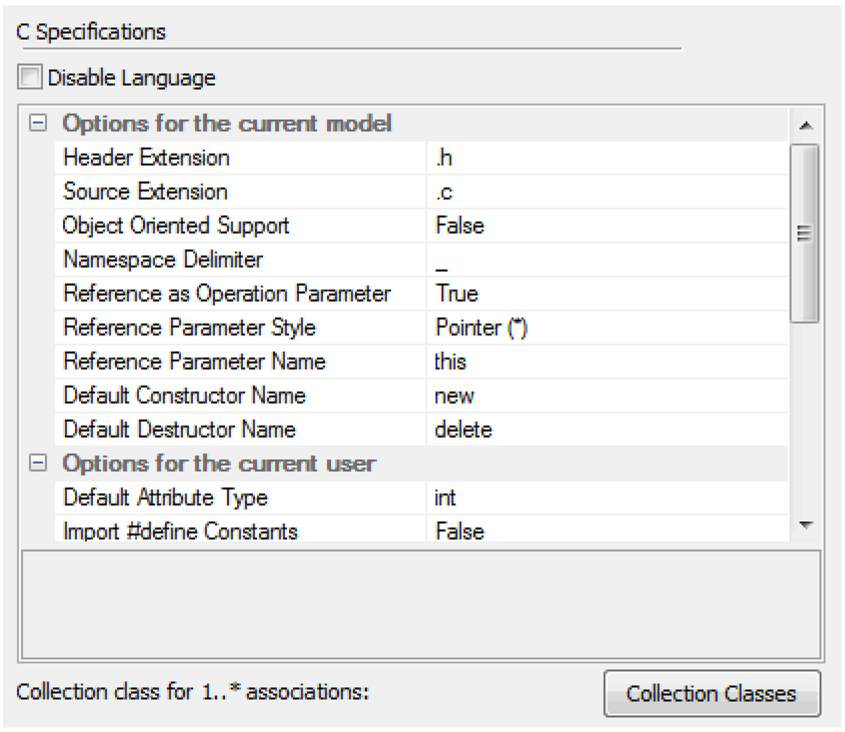

Ada
Disable Language
Options for the current user
Collection class for 1..* associations:
Collection Classes

Default Attribute Type	
Generate Namespaces	True
Default Source Directory	
Import File Extensions	.ads;
Editor	
Use Class Name for Tagged Record	True
Alternate Tagged Record Name^	Typ
Define Reference for Tagged Record	False
Reference Type Name^	Ref
Ref Parameter Style	Access
Ignore Reference parameter name ...	False
Ref Parameter Name^	this

1.4.6.3 C Options

Configure options for C code generation using the C Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | C** menu option). The options you can specify include:

- Support for Object Oriented coding
- Default file extensions (header and source)
- Default source directory
- Editor for C code
- Path that Enterprise Architect uses to search for the implementation file; the first path in the list is the default path when generating.



Options for the current model

Header Extension	.h
Source Extension	.c
Object Oriented Support	False
Namespace Delimiter	_
Reference as Operation Parameter	True
Reference Parameter Style	Pointer (*)
Reference Parameter Name	this
Default Constructor Name	new
Default Destructor Name	delete

Options for the current user

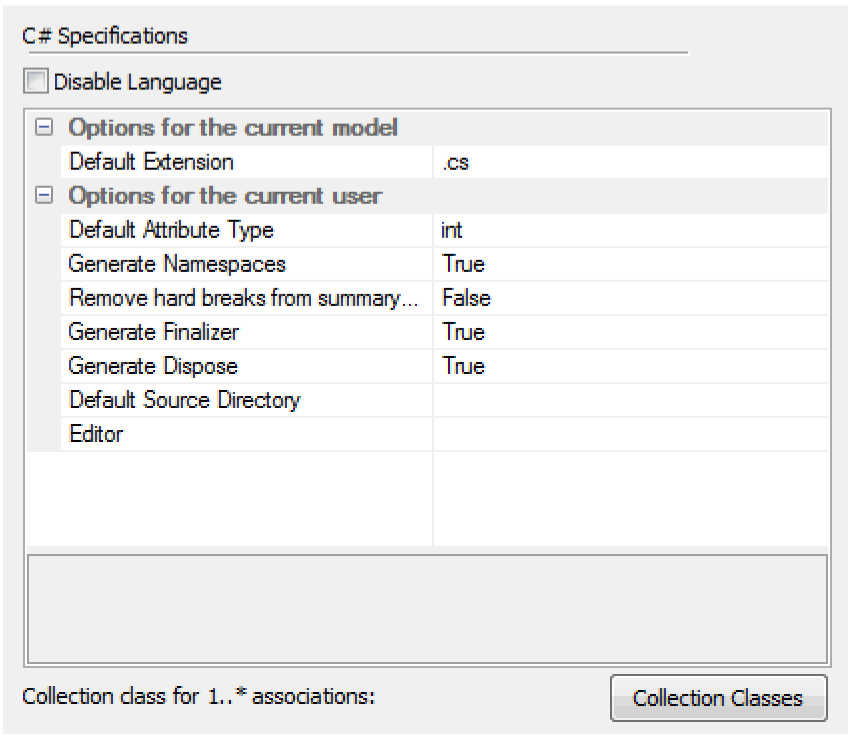
Default Attribute Type	int
Import #define Constants	False

Collection class for 1..* associations:

1.4.6.4 C# Options

Configure options for C# code generation using the C# Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | C#** menu option). The options you can specify include the default:

- File extension
- 'Get' prefix
- 'Set' prefix
- Directory for opening and saving C# source code.



C# Specifications

☐ Disable Language

⊟ Options for the current model	
Default Extension	.cs
⊟ Options for the current user	
Default Attribute Type	int
Generate Namespaces	True
Remove hard breaks from summary...	False
Generate Finalizer	True
Generate Dispose	True
Default Source Directory	
Editor	

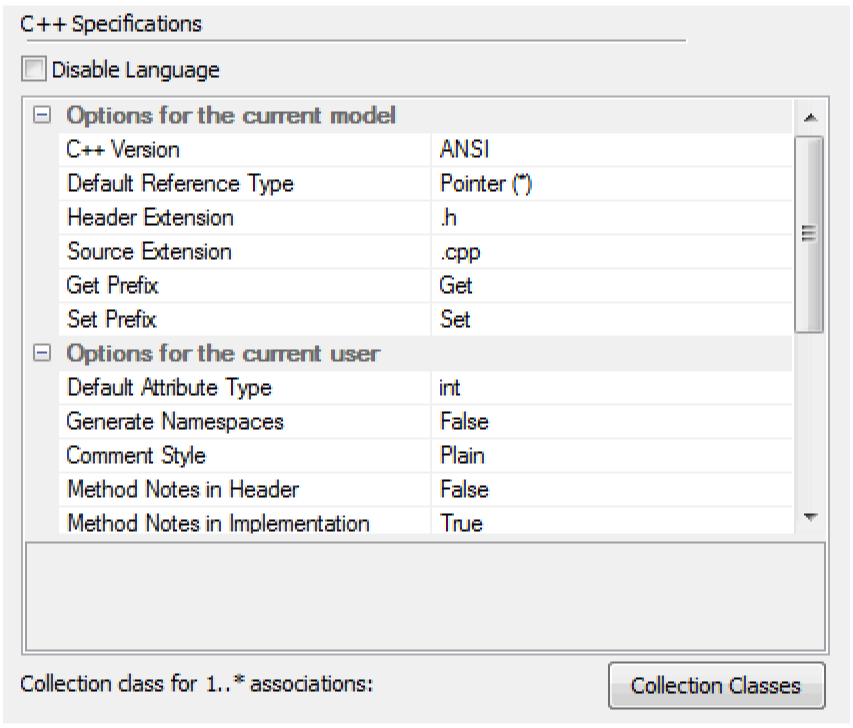
Collection class for 1..* associations:

1.4.6.5 C++ Options

Configure options for C++ code generation using the C++ Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | C++** menu option). The options you can specify include:

- The version of C++ to generate; this controls the set of templates used and how properties are created
- The default reference type used when a type is specified by reference
- The default file extensions
- Default Get/Set prefixes
- Default source directory
- The path that Enterprise Architect uses to search for the implementation file. The first path in the list is the default path when generating new implementation files and parsing existing files; if you add further directories, Enterprise Architect also searches these when parsing existing files.

  For example, you have a directory *inc* that contains all of your headers, while the source code is mixed through directories *src*, *srca*, and *srcb*. You therefore set the **Source Path** option to **../src/;../srca/;../srcb/**. This ensures that new implementation files are generated into *src*, but when parsing existing files Enterprise Architect looks in all three source directories (but never in the *inc* directory). You must still ensure that the implementation file name matches the header file name, and that the file extension matches the extension specified in the options. If these conditions are not met, Enterprise Architect cannot handle that code.


C++ Specifications

Disable Language

Options for the current model	
C++ Version	ANSI
Default Reference Type	Pointer (*)
Header Extension	.h
Source Extension	.cpp
Get Prefix	Get
Set Prefix	Set
Options for the current user	
Default Attribute Type	int
Generate Namespaces	False
Comment Style	Plain
Method Notes in Header	False
Method Notes in Implementation	True

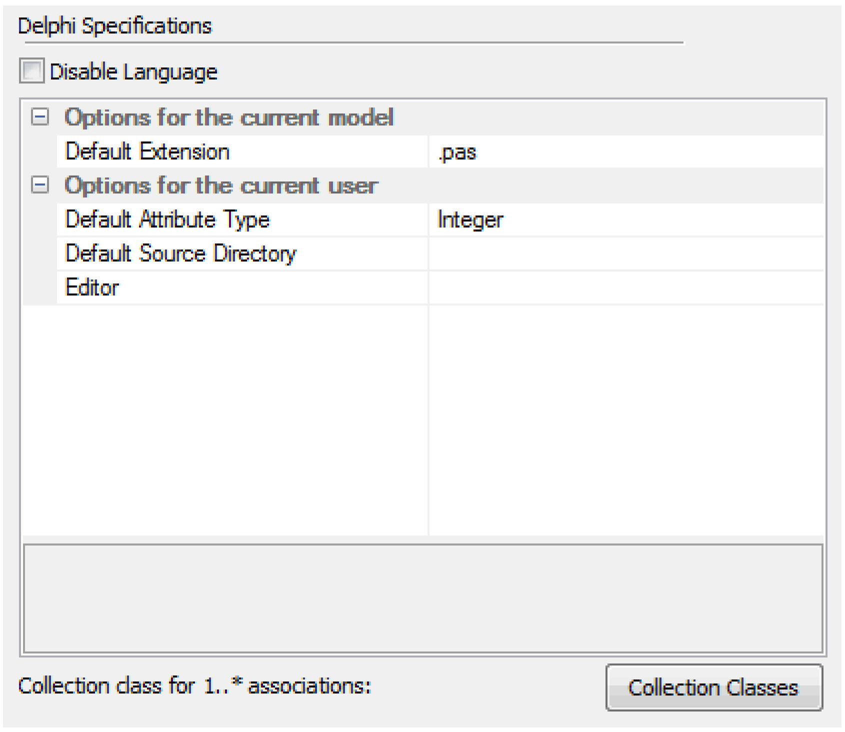
Collection class for 1..* associations:

1.4.6.6 Delphi Options

Configure options for Delphi code generation using the Delphi Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | Delphi** menu option). The options you can specify include the:

- Default file extension
- Default source directory

You can also set a default directory for opening and saving Delphi source code.


Delphi Specifications

☐ Disable Language

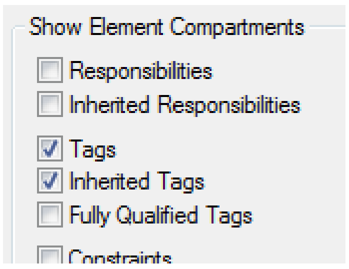
Options for the current model	
Default Extension	.pas
Options for the current user	
Default Attribute Type	Integer
Default Source Directory	
Editor	

Collection class for 1..* associations: Collection Classes

You should also set Delphi properties 48 within each Class element.

1.4.6.6.1 *Delphi Properties*

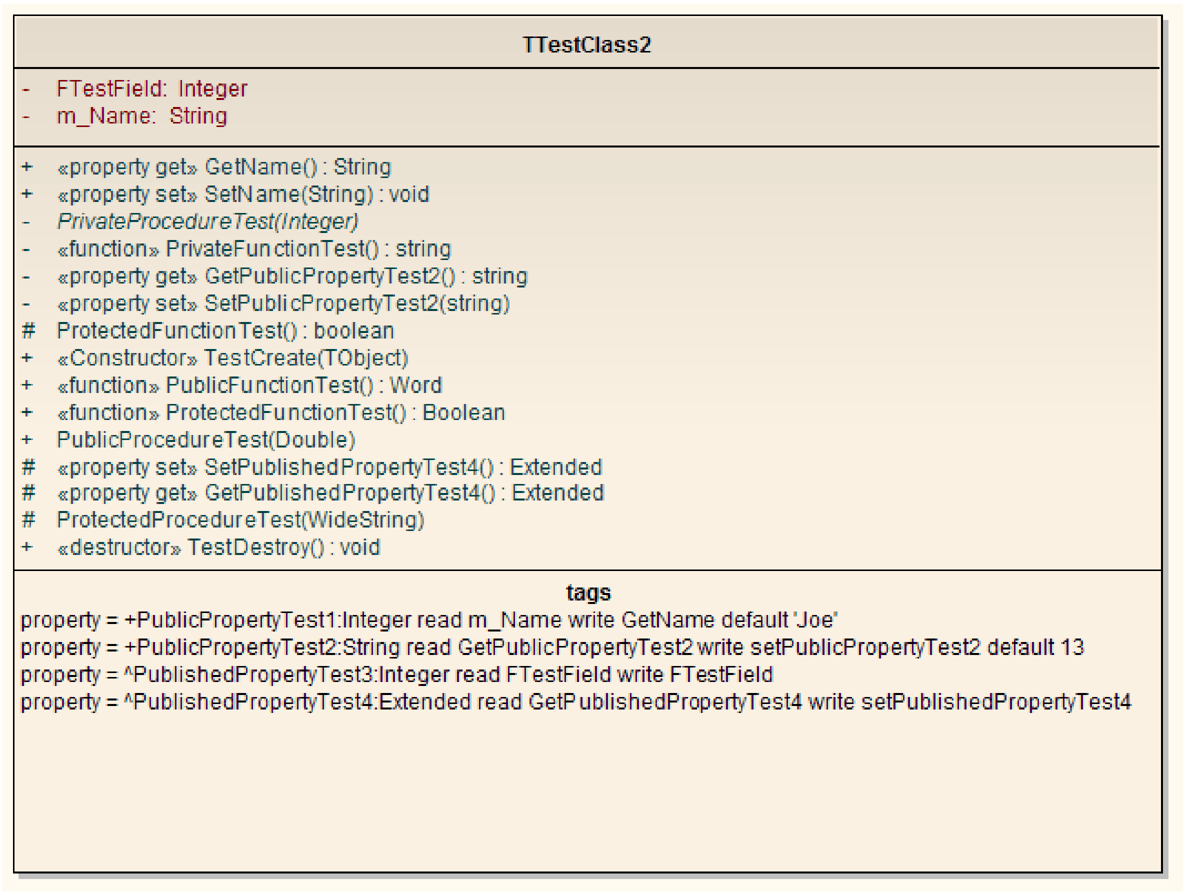
Enterprise Architect has comprehensive support for Delphi properties. These are implemented as Tagged Values, with a specialized property editor to help create and modify Class properties. The Class image below illustrates the appearance of a Delphi Class that has had properties added to it. These are stored as Tagged Values, and by using the **Feature Visibility** element context menu option, you can display the 'tags' compartment that contains the properties. Imported Delphi Classes with properties have this feature automatically made visible for your convenience.


Show Element Compartments

☐ Responsibilities
☐ Inherited Responsibilities
☑ Tags
☑ Inherited Tags
☐ Fully Qualified Tags
☐ Constraints

Note:

When you use the Create Property dialog from the Attribute screen, Enterprise Architect generates a pair of Get and Set functions, together with the required property definition as Tagged Values. You can manually edit these Tagged Values if required.



TTestClass2

- - FTestField: Integer
- - m_Name: String

```
+  «property get» GetName() : String
+  «property set» SetName(String) : void
-  PrivateProcedureTest(Integer)
-  «function» PrivateFunctionTest() : string
-  «property get» GetPublicPropertyTest2() : string
-  «property set» SetPublicPropertyTest2(string)
#  ProtectedFunctionTest() : boolean
+  «Constructor» TestCreate(TObject)
+  «function» PublicFunctionTest() : Word
+  «function» ProtectedFunctionTest() : Boolean
+  PublicProcedureTest(Double)
#  «property set» SetPublishedPropertyTest4() : Extended
#  «property get» GetPublishedPropertyTest4() : Extended
#  ProtectedProcedureTest(WideString)
+  «destructor» TestDestroy() : void
```

tags

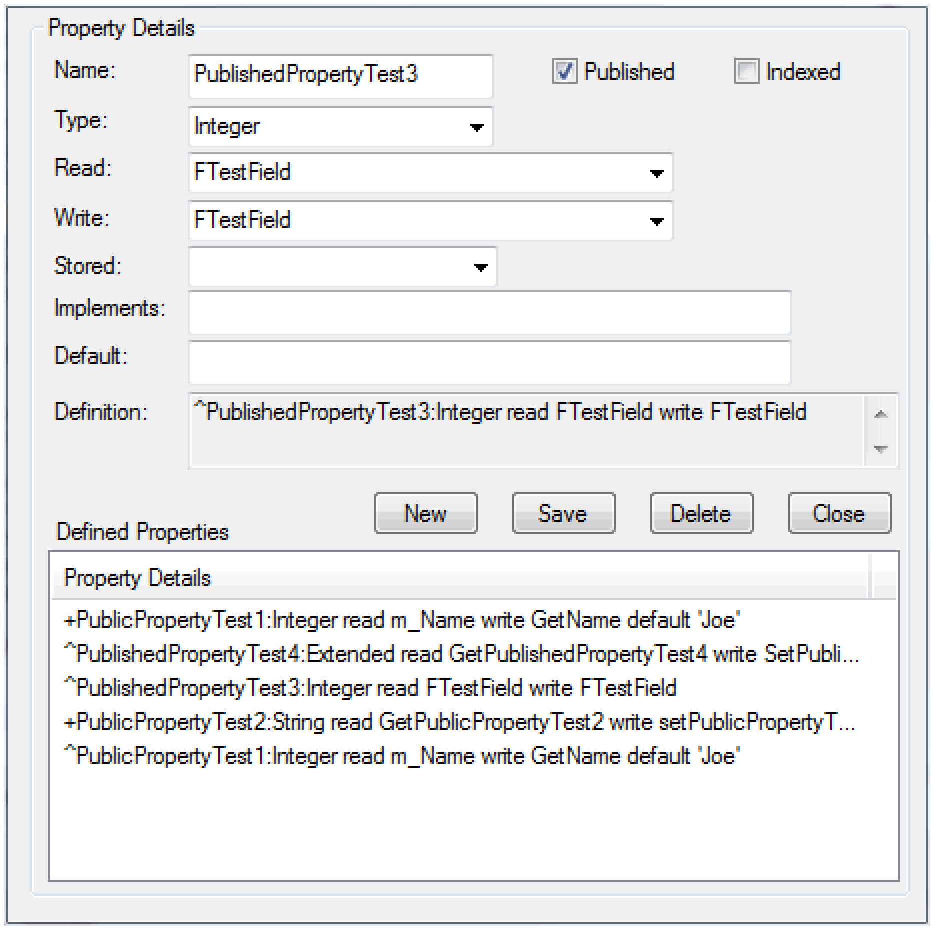
```
property = +PublicPropertyTest1:Integer read m_Name write GetName default 'Joe'
property = +PublicPropertyTest2:String read GetPublicPropertyTest2 write setPublicPropertyTest2 default 13
property = ^PublishedPropertyTest3:Integer read FTestField write FTestField
property = ^PublishedPropertyTest4:Extended read GetPublishedPropertyTest4 write setPublishedPropertyTest4
```

To manually activate the property editor

1. Ensure the Class you have selected has the code generation language set to Delphi
2. Right-click on the Class and select the **Delphi Properties** context menu option to open the editor.

The Delphi Properties editor enables you to build properties in a simple and straightforward manner. From here you can:

- Change the name and scope (only **Public** and **Published** are currently supported)
- Change the property type (the drop-down list includes all defined Classes in the project)
- Set the **Read** and **Write** information (the drop-down lists have all the attributes and operations from the current Class; you can also enter free text)
- Set **Stored** to **True** or **False**
- Set the **Implements** information
- Set the **Default** value, if one exists.


Property Details

Name: PublishedPropertyTest3 [✓ Published] Indexed

Type: Integer

Read: FTestField

Write: FTestField

Stored:

Implements:

Default:

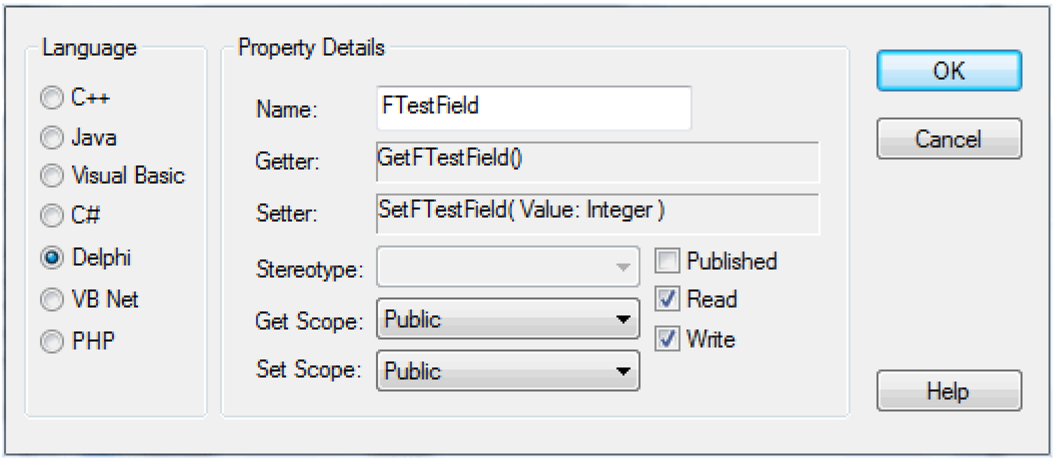
Definition: ^PublishedPropertyTest3:Integer read FTestField write FTestField

Defined Properties

Property Details
+PublicPropertyTest1:Integer read m_Name write GetName default 'Joe'
^PublishedPropertyTest4:Extended read GetPublishedPropertyTest4 write SetPubli...
^PublishedPropertyTest3:Integer read FTestField write FTestField
+PublicPropertyTest2:String read GetPublicPropertyTest2 write setPublicPropertyT...
^PublicPropertyTest1:Integer read m_Name write GetName default 'Joe'

Notes:

- Public properties are displayed with a '**+**' symbol prefix and published with a '**^**'.
- When creating a property in the Create Property Implementation dialog (accessed through the Attributes dialog), you can set the scope to **Published** if the property type is Delphi - see the example below.


Language: C++ Java Visual Basic C# [● Delphi] VB Net PHP

Property Details

Name: FTestField

Getter: GetFTestField()

Setter: SetFTestField(Value: Integer)

Stereotype: Published [✓ Read] [✓ Write]

Get Scope: Public

Set Scope: Public

Limitations

- Only **Public** and **Published** are supported
- If you change the name of a property and forward engineer, a new property is added, but the old one must

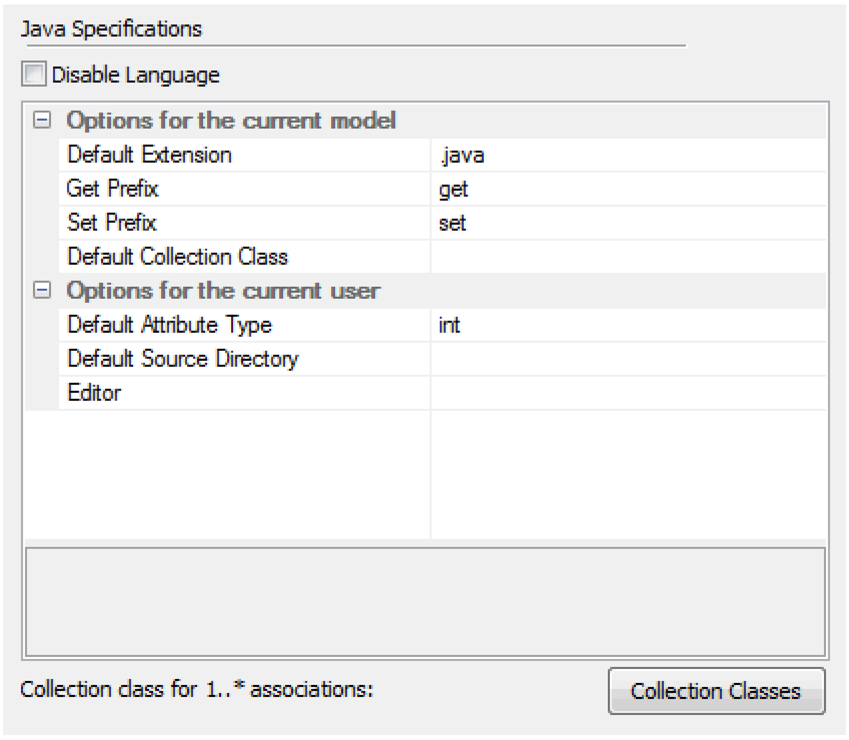
be manually deleted from the source file.

1.4.6.7 Java Options

Configure options for Java code generation using the Java Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | Java** menu option). The options you can specify include the:

- Default file extension
- Default 'Get' prefix
- Default 'Set' prefix

You can also set a default directory for opening and saving Java source code.



Java Specifications

☐ Disable Language

⊟ Options for the current model	
Default Extension	.java
Get Prefix	get
Set Prefix	set
Default Collection Class	
⊟ Options for the current user	
Default Attribute Type	int
Default Source Directory	
Editor	

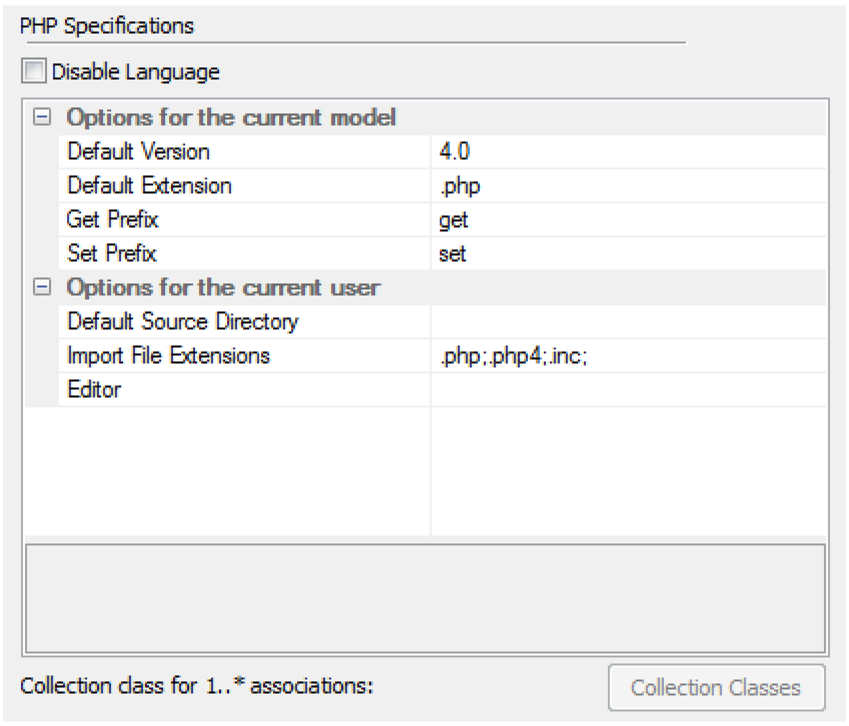
Collection class for 1..* associations:

1.4.6.8 PHP Options

Configure options for PHP code generation using the PHP Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | PHP** menu option). The options you can specify include the:

- Default source extension - specify the extension to be used when creating files for PHP source
- Default import extension - a semi-colon separated list of extensions to look at when doing a directory code import 8 for PHP
- Default PHP version - the version of PHP to generate.

You can also set a default directory for opening and saving PHP source code.


PHP Specifications

Disable Language

Options for the current model	
Default Version	4.0
Default Extension	.php
Get Prefix	get
Set Prefix	set
Options for the current user	
Default Source Directory	
Import File Extensions	.php;.php4;.inc;
Editor	

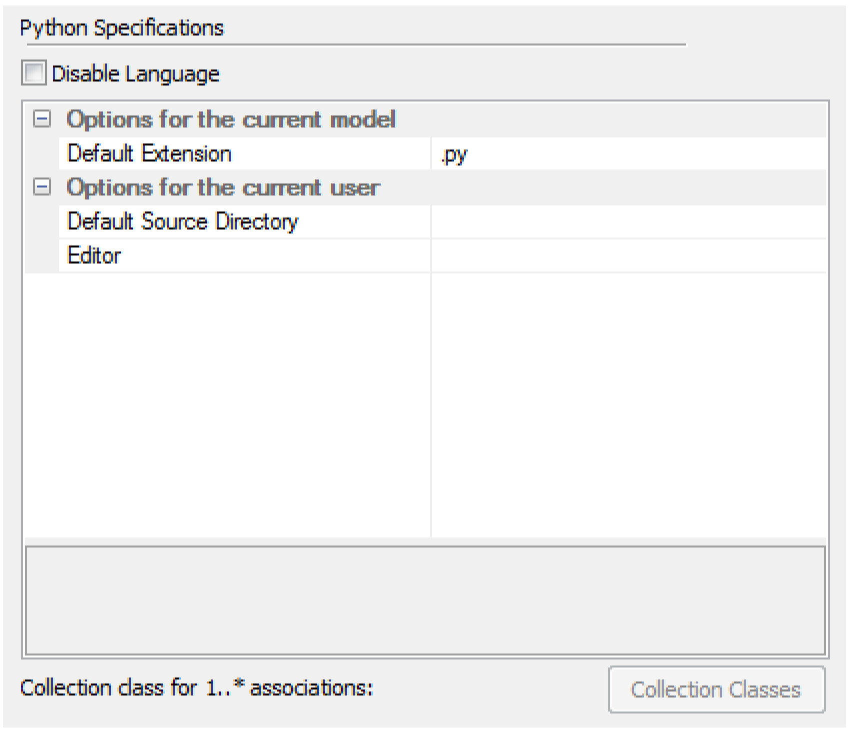
Collection class for 1..* associations:

1.4.6.9 Python Options

Configure options for Python code generation using the Python Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | Python** menu option). The options you can specify include the:

- Default file extension(s)
- Default source directory.

You can also set the editor for Python code.



Python Specifications

☐ Disable Language

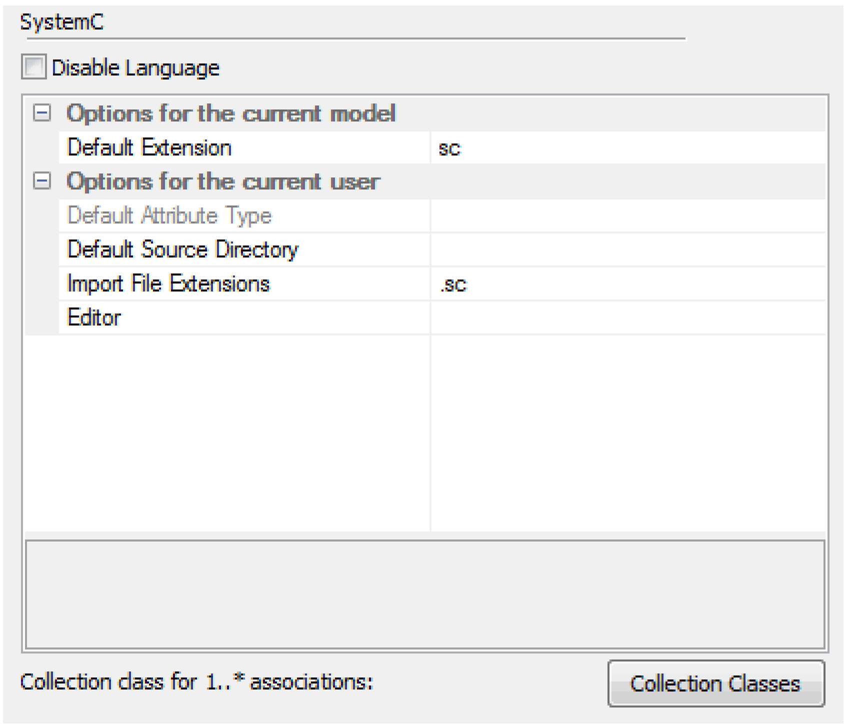
⊟ Options for the current model	
Default Extension	.py
⊟ Options for the current user	
Default Source Directory	
Editor	

Collection class for 1..* associations:

1.4.6.10 SystemC Options

Configure options for SystemC code generation using the SystemC page of the Options dialog (select the **Tools | Options | Source Code Engineering | SystemC** menu option). The options you can specify include the:

- Default file extension(s)
- Default source directory
- Editor for changing code.



SystemC

☐ Disable Language

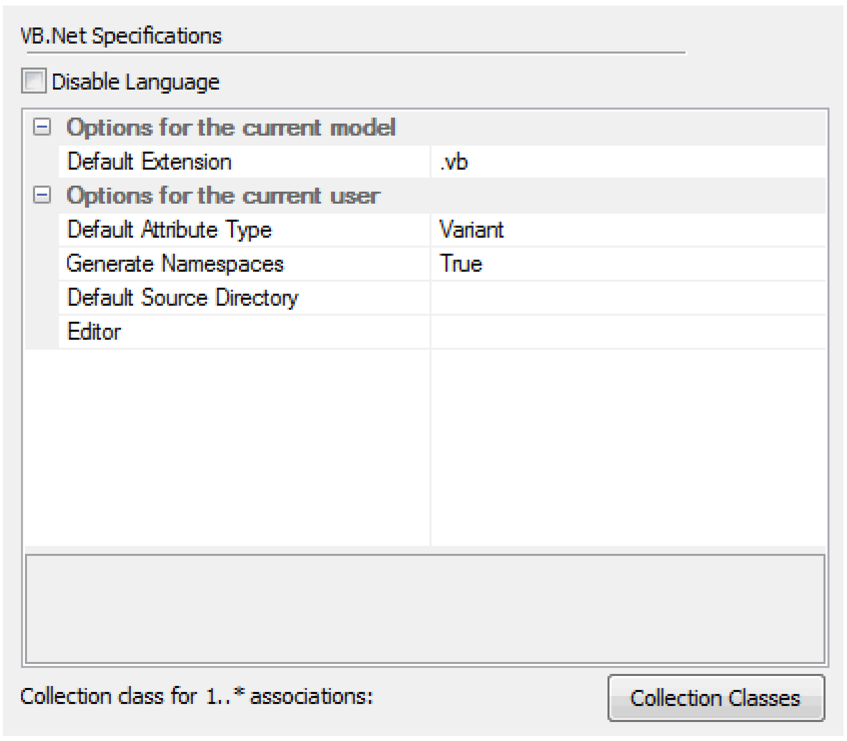
Options for the current model	
Default Extension	sc
Options for the current user	
Default Attribute Type	
Default Source Directory	
Import File Extensions	.sc
Editor	

Collection class for 1..* associations:

1.4.6.11 VB.Net Options

Configure options for VB.Net code generation using the VB.Net Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | VB.Net** menu option). The options you can specify include the:

- Default file extension
- Default source directory.



VB.Net Specifications

Disable Language

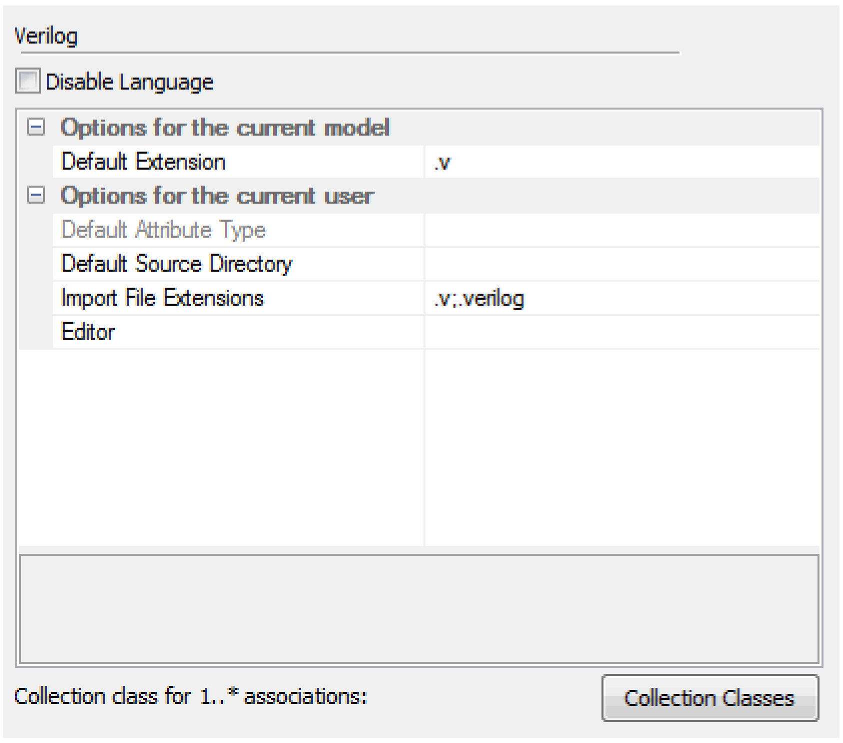
Options for the current model	
Default Extension	.vb
Options for the current user	
Default Attribute Type	Variant
Generate Namespaces	True
Default Source Directory	
Editor	

Collection class for 1..* associations:

1.4.6.12 Verilog Options

Configure options for Verilog code generation using the Verilog page of the Options dialog (select the **Tools | Options | Source Code Engineering | Verilog** menu option). The options you can specify include the:

- Default file extension(s)
- Default source directory
- Editor for changing code.


Verilog

☐ Disable Language

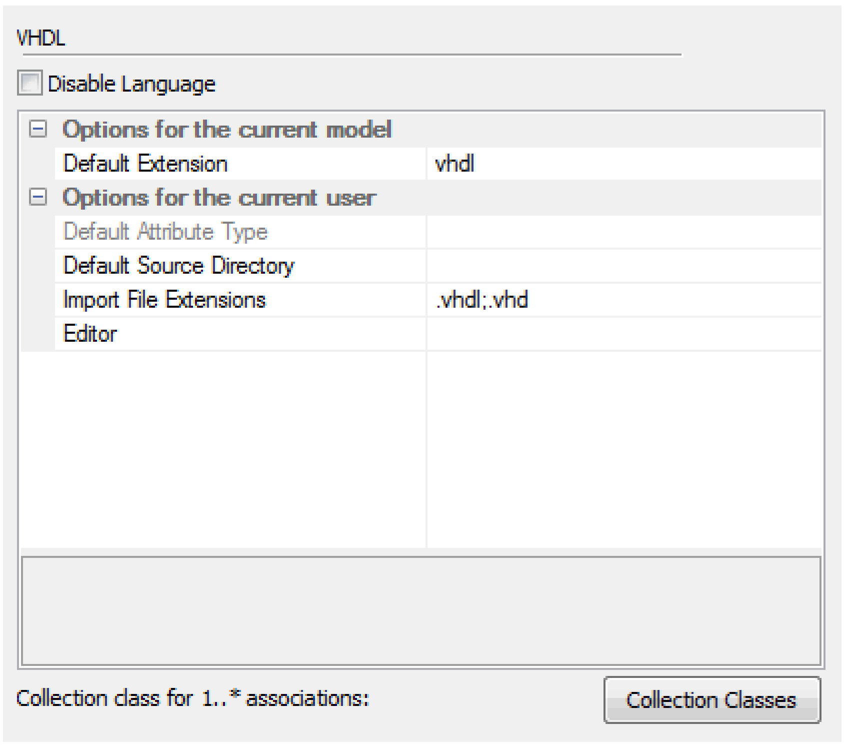
⊟ **Options for the current model**	
Default Extension	.v
⊟ **Options for the current user**	
Default Attribute Type	
Default Source Directory	
Import File Extensions	.v;.verilog
Editor	

Collection class for 1..* associations:

1.4.6.13 VHDL Options

Configure options for VHDL code generation using the VHDL page of the Options dialog (select the **Tools | Options | Source Code Engineering | VHDL** menu option). The options you can specify include the:

- Default file extension(s)
- Default source directory
- Editor for changing code.


VHDL

Disable Language

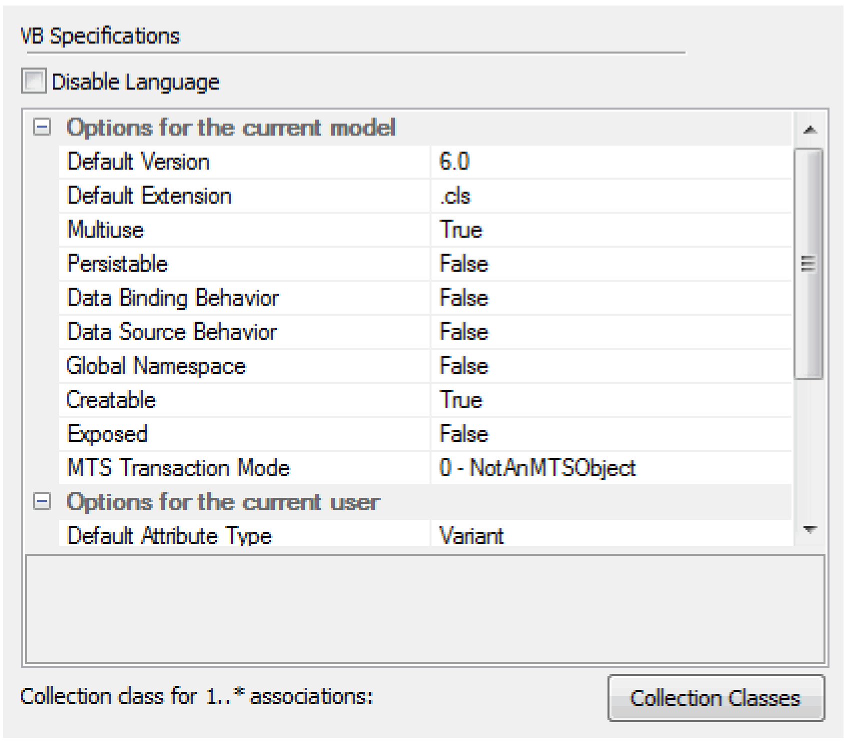
Options for the current model	
Default Extension	vhdl
Options for the current user	
Default Attribute Type	
Default Source Directory	
Import File Extensions	.vhdl;.vhd
Editor	

Collection class for 1..* associations:

1.4.6.14 Visual Basic Options

Configure options for Visual Basic code generation using the VB Specifications page of the Options dialog (select the **Tools | Options | Source Code Engineering | Visual Basic** menu option). The options you can specify include the:

- Default file extension when reading/writing
- Default Visual Basic version
- MTS transaction mode for MTS objects
- Multi use (true or false)
- Persistable
- Data binding
- Global namespace
- Exposed
- Data source behavior
- Creatable.



Options for the current model	
Default Version	6.0
Default Extension	.cls
Multiuse	True
Persistable	False
Data Binding Behavior	False
Data Source Behavior	False
Global Namespace	False
Creatable	True
Exposed	False
MTS Transaction Mode	0 - NotAnMTSObject
Options for the current user	
Default Attribute Type	Variant

Collection class for 1..* associations:

1.4.6.15 MDG Technology Language Options

If you have loaded an MDG Technology that specifies a code module into your *Sparx Systems > EA > MDG Technologies* folder (see the *MDG Technologies in SDK* section of *SDK For Enterprise Architect*), the language is included in the Source Code Engineering list on the Options dialog. The language is only listed on the Options dialog if an MDG Technology file actually uses it in your model.

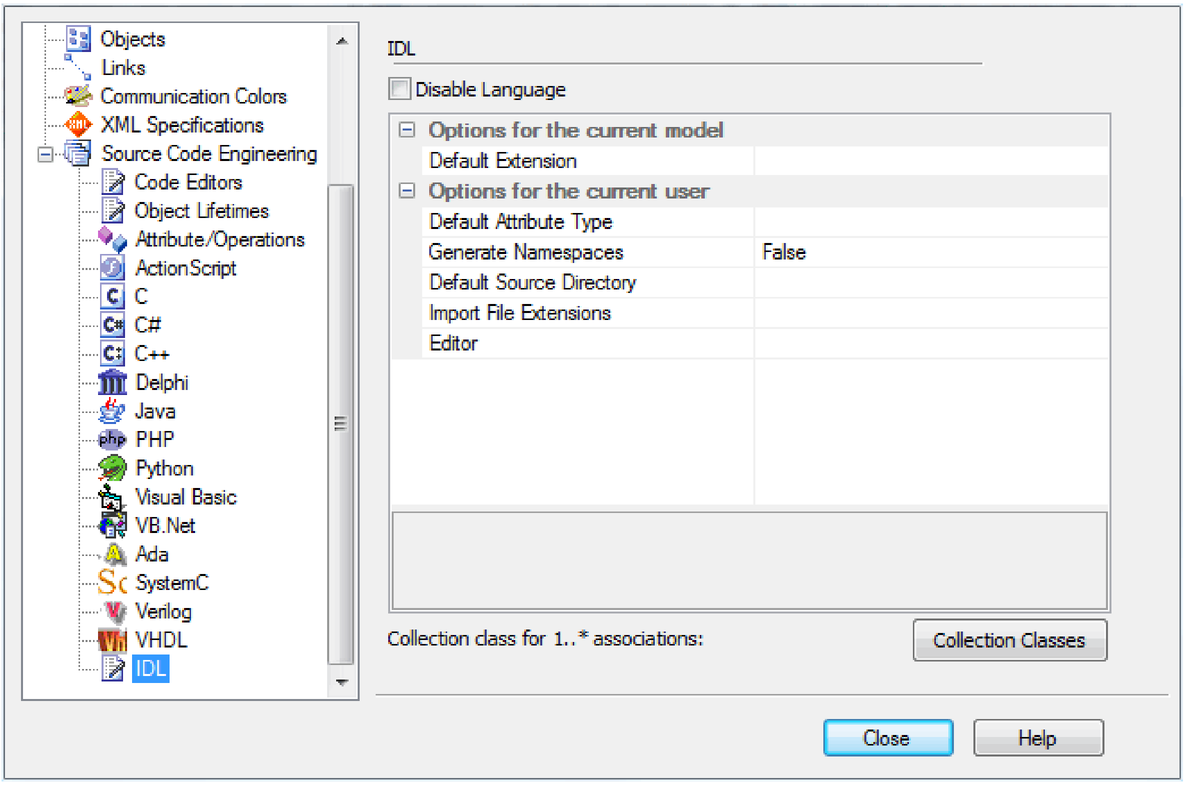
The options for each language are based on what is defined in the technology code module, but are limited to the following:

- **Default Extension**
  - Default extension for generated source files
  - Shown if the option is in the technology
  - Saved per project.
- **Import File Extensions**
  - Default folder to import source files from
  - Shown if there is a grammar set in the technology
  - Saved once for all projects.
- **Generate Namespaces**
  - Option to generate namespaces or not
  - Shown if the technology supports namespaces
  - Saved once for all projects.
- **Default Source Directory**
  - The default directory to save generated source files
  - Always shown
  - Saved once for all projects.
- **Editor**
  - The editor that is loaded to edit the source files
  - Always shown
  - Saved once for all projects.

- **Att Type**
  - Default type for attributes
  - Always shown
  - Saved once for all projects.

These options are set in the technology inside the <*CodeOptions*> tag of a code module, as follows:

```
<CodeOption name="DefaultExtension">.rb</CodeOption>
```



IDL

☐ Disable Language

Options for the current model	
Default Extension	
Options for the current user	
Default Attribute Type	
Generate Namespaces	False
Default Source Directory	
Import File Extensions	
Editor	

Collection class for 1..* associations:

1.4.6.16 Reset Options

Enterprise Architect stores some of the options for a Class when it is first created. Some are global; for example *$LinkClass* is stored when you first create the Class, so it won't automatically pick up the global change in the Options dialog in existing Classes. You must modify the options for the existing Class.

Modify Options for Single Class

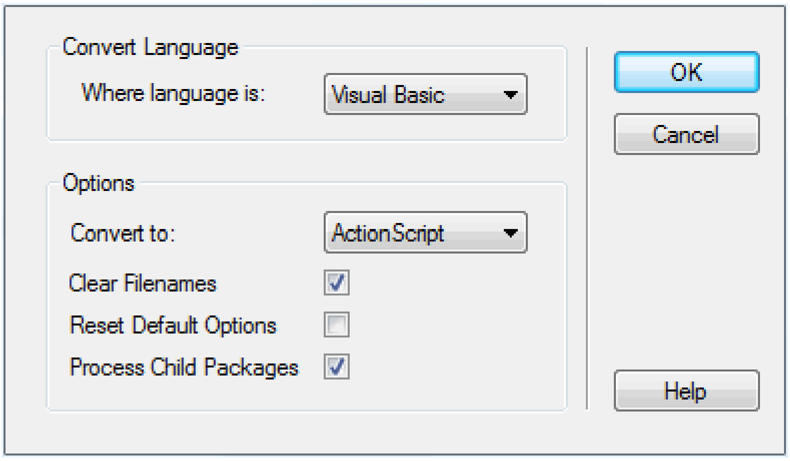
To modify options for a single Class, follow the steps below:

1. Right-click on the Class to change, and select the **Generate Code** menu option from the context menu. The Generate Code dialog displays.
2. Click on the **Advanced** button. The Object Options dialog displays.
3. Click on the **Attributes/Operations** button.
4. Change the options, and click on the **Close** button to apply changes.

Modify Options for All Classes

To modify options for all Classes within a package, follow the steps below:

1. Right-click on the package in the Project Browser. The context menu displays.
2. Select the **Code Engineering | Reset Options for this Package** menu option. The Manage Code Generation dialog displays.


Convert Language

Where language is: Visual Basic

Options

Convert to: ActionScript

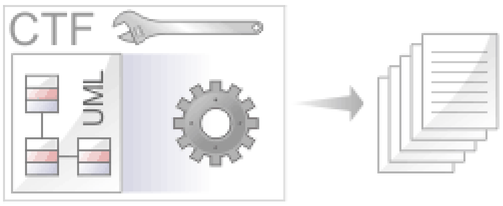
Clear Filenames [✓]

Reset Default Options [ ]

Process Child Packages [✓]

3. Reset the required defaults for each existing Class.
4. Click on the **OK** button to apply changes.

1.5 Code Template Framework



The Code Template Framework (CTF) is used during forward engineering of UML models. The CTF enables you to:

- Generate source code from UML models
- Customize the way in which Enterprise Architect generates source code
- Forward engineer languages not specifically supported by Enterprise Architect.

The CTF consists of:

- Default Code Templates 61 which are built into Enterprise Architect for forward engineering supported languages
- A Code Template Editor 64 for creating and maintaining user-defined Code Templates (also see *SDK for Enterprise Architect*)
- Code templates to synchronize code 66.

1.5.1 Code Templates

Code templates enable you to customize code generation of existing languages. For example:

- Modify the file headers created when generating new files
- Change the style of the generated code (such as indenting or brace position) to match the required coding standards
- Handle particular stereotypes to generate things like specialized method bodies and extra methods.

They also enable you to add code generation of entirely new languages that Enterprise Architect would otherwise not be able to handle. In this situation it is most useful to combine code templates with an MDG technology file that includes the datatypes, and options for default file extensions. See the *Create MDG Technologies* topic in *SDK for Enterprise Architect*.

Enterprise Architect's base code templates 62 specify the transformation from UML elements to the various parts of a given programming language. The templates are written as plain text with a syntax that shares some aspects of both mark-up languages and scripting languages. A simple example of a template used by Enterprise Architect is the 'Class template'. It is used to generate source code from a UML Class:

```
%ClassNotes%
%ClassDeclaration%
%ClassBody%
```

The above template simply refers to three other templates, namely *ClassNotes*, *ClassDeclaration* and *ClassBody*. The enclosing percent (%) signs indicate a *macro*. Code Templates consist of various types of macros, each resulting in a substitution in the generated output. For a language such as C++, the result of processing the above template might be:

```
/**
 * This is an example class note generated using code templates
 * @author Sparx Systems
 */
class ClassA: public ClassB
{
...
}
```

Execution of Code Templates

A reference to a template (such as the *%ClassNotes%* macro, from our example above) results in the execution of that template.

Each template is designed for use with a particular element. For example the *ClassNotes* template is to be used with UML Class elements.

The element that is currently being generated is said to be *in scope*. If the element in scope is stereotyped Enterprise Architect looks for a template that has been defined for that stereotype. If a match is found, the specialized template is executed. Otherwise the default implementation of the base template is used.

Templates are processed sequentially, line by line, replacing each macro with its underlying text value from the model.

1.5.1.1 Base Templates

The Code Template Framework consists of a number of base templates. Each base template transforms particular aspects of the UML to corresponding parts of object-oriented languages.

The following table lists and briefly describes the base templates used in the CTF.

Template	Description
Attribute	A top-level template to generate member variables from UML attributes.
Attribute Declaration	Used by the *Attribute* template to generate a member variable declaration.
Attribute Notes	Used by the *Attribute* template to generate member variable notes.
Class	A top-level template for generating Classes from UML Classes.
Class Base	Used by the *Class* template to generate a base Class name in the inheritance list of a derived Class, where the base Class doesn't exist in the model.
Class Body	Used by the *Class* template to generate the body of a Class.
Class Declaration	Used by the *Class* template to generate the declaration of a Class.
Class Interface	Used by the *Class* template to generate an interface name in the inheritance list of a derived Class, where the interface doesn't exist in the model.
Class Notes	Used by the *Class* template to generate the Class notes.
File	A top-level template for generating the source file. For languages such as C++, this corresponds to the header file.
Import Section	Used in the *File* template to generate external dependencies.
Linked Attribute	A top-level template for generating attributes derived from UML Associations.
Linked Attribute Notes	Used by the *Linked Attribute* template to generate the attribute notes.
Linked Attribute Declaration	Used by the *Linked Attribute* template to generate the attribute declaration.
Linked Class Base	Used by the *Class* template to generate a base Class name in the inheritance list of a derived Class, for a Class element in the model that is a parent of the current Class.
Linked Class Interface	Used by the *Class* template to generate an Interface name in the inheritance list of a derived Class, for an Interface element in the model that is a parent of the current Class.
Namespace	A top-level template for generating namespaces from UML packages. (Although not all languages have namespaces, this template can be used to generate an equivalent construct, such as packages in Java.)
Namespace Body	Used by the *Namespace* template to generate the body of a namespace.
Namespace Declaration	Used by the *Namespace* template to generate the namespace declaration.

Template	Description
Operation	A top-level template for generating operations from a UML Class's operations.
Operation Body	Used by the *Operation* template to generate the body of a UML operation.
Operation Declaration	Used by the *Operation* template to generate the operation declaration.
Operation Notes	Used by the *Operation* template to generate documentation for an operation.
Parameter	Used by the *Operation Declaration* template to generate parameters.

The second table lists templates used for generating code for languages that have separate interface and implementation sections.

Template	Description
Class Impl	A top-level template for generating the implementation of a Class.
Class Body Impl	Used by the *Class Impl* template to generate the implementation of Class members.
File Impl	A top-level template for generating the implementation file.
File Notes Impl	Used by the *File Impl* template to generate notes in the source file.
Import Section Impl	Used by the *File Impl* template to generate external dependencies.
Operation Impl	A top-level template for generating operations from a UML Class's operations.
Operation Body Impl	Used by the *Operation* template to generate the body of a UML operation.
Operation Declaration Impl	Used by the *Operation* template to generate the operation declaration.
Operation Notes Impl	Used by the *Operation* template to generate documentation for an operation.

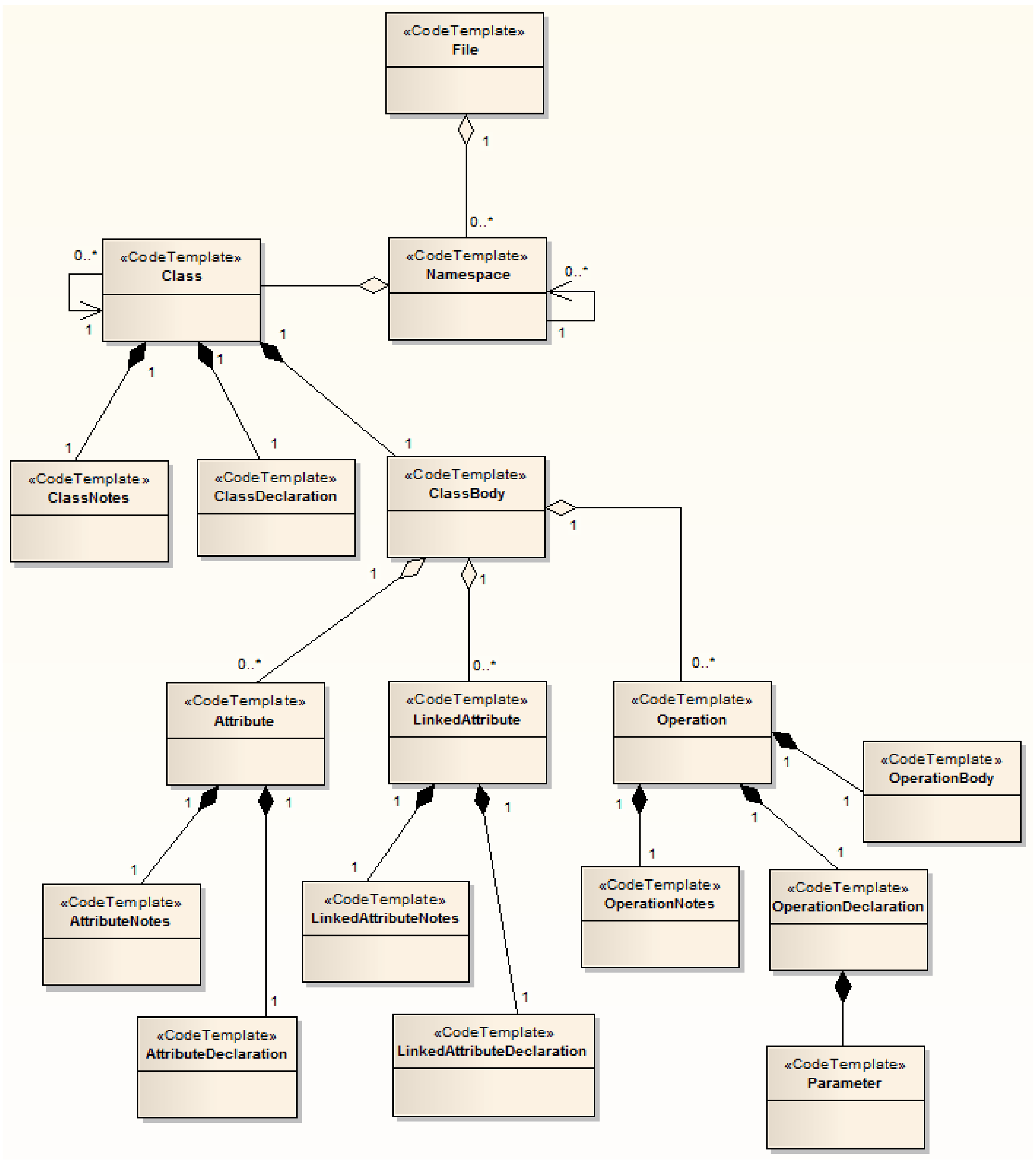
The base templates form a hierarchy, which varies slightly across different programming languages. A typical template hierarchy relevant to a language like C# or Java (which do not have header files) is shown in the example diagram below. In this diagram the templates are modeled as Classes (in reality they are just plain text). This hierarchy would be slightly more complicated for languages like C++ and Delphi, which have separate implementation templates.

Each of the base templates must be specialized to be of use in code engineering. In particular, each template is specialized for the supported languages (or 'products'). For example, there is a *ClassBody* template defined for C++, another for C#, another for Java, and so on. By specializing the templates, you can tailor the code generated for the corresponding UML entity.

Once the base templates are specialized for a given language, they can be further specialized based on:

- A Class's stereotype
- A feature's stereotype (where the feature can be an operation or attribute)

This type of specialization enables, for example, a C# operation that is stereotyped as «property» to have a different *Operation Body* template from an ordinary operation. The *Operation Body* template can then be specialized further, based on the Class stereotype.


«CodeTemplate» File
«CodeTemplate» Namespace
«CodeTemplate» Class
«CodeTemplate» ClassNotes
«CodeTemplate» ClassDeclaration
«CodeTemplate» ClassBody
«CodeTemplate» Attribute
«CodeTemplate» LinkedAttribute
«CodeTemplate» Operation
«CodeTemplate» OperationBody
«CodeTemplate» AttributeNotes
«CodeTemplate» LinkedAttributeNotes
«CodeTemplate» OperationNotes
«CodeTemplate» OperationDeclaration
«CodeTemplate» AttributeDeclaration
«CodeTemplate» LinkedAttributeDeclaration
«CodeTemplate» Parameter
1
0..*

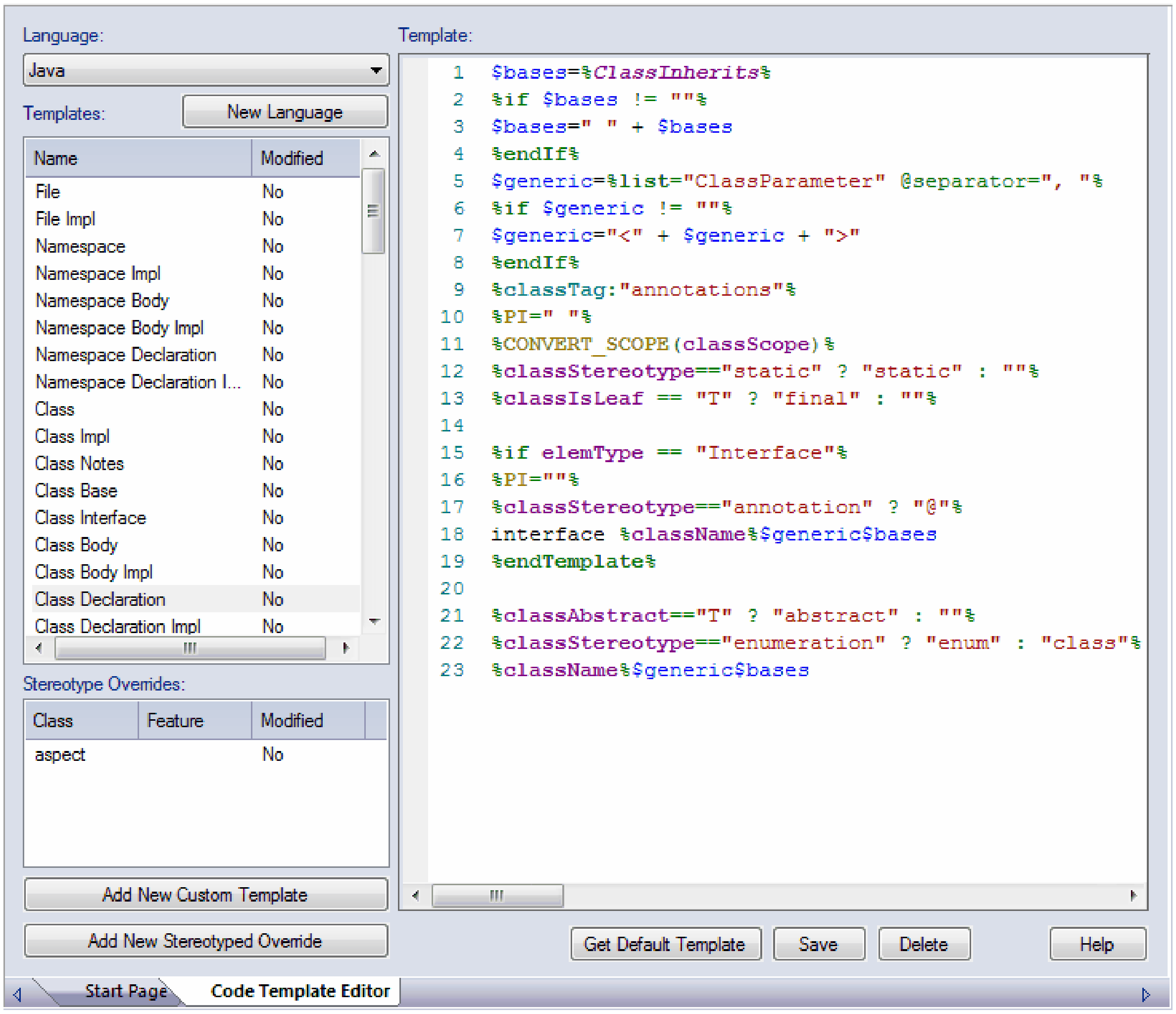
Note:

The above Class Model shows the hierarchy of Code Generation templates for a language such as C# or Java. The Aggregation connectors denote references between templates.

1.5.2 The Code Template Editor

The Code Template Editor provides the facilities of the *Common Code Editor*, including intellisense for the various macros. For more information on intellisense and the Common Code Editor, see the *Code Editors* topic in *Using Enterprise Architect - UML Modeling Tool*.

To access the Code Template Editor window, select the **Settings | Code Generation Templates** menu option.



Language: Java

Templates:

Name	Modified
File	No
File Impl	No
Namespace	No
Namespace Impl	No
Namespace Body	No
Namespace Body Impl	No
Namespace Declaration	No
Namespace Declaration I...	No
Class	No
Class Impl	No
Class Notes	No
Class Base	No
Class Interface	No
Class Body	No
Class Body Impl	No
Class Declaration	No
Class Declaration Impl	No

Stereotype Overrides:

Class	Feature	Modified
aspect		No

Template:

```
$bases=%ClassInherits%
%if $bases != ""%
$bases=" " + $bases
%endIf%
$generic=%list="ClassParameter" @separator=", "%
%if $generic != ""%
$generic="<" + $generic + ">"
%endIf%
%classTag:"annotations"%
%PI=" "%
%CONVERT_SCOPE(classScope)%
%classStereotype=="static" ? "static" : ""%
%classIsLeaf == "T" ? "final" : ""%

%if elemType == "Interface"%
%PI=""%
%classStereotype=="annotation" ? "@"%
interface %className%$generic$bases
%endTemplate%

%classAbstract=="T" ? "abstract" : ""%
%classStereotype=="enumeration" ? "enum" : "class"%
%className%$generic$bases
```

Option	Use to
Language	Select the programming language.
New Language	Display the Programming Languages Datatypes dialog (see the *Reference Data* topic in *UML Model Management*), which enables you to include programming languages other than those supported for Enterprise Architect, for which to create or edit code templates.
Template	Display the contents of the active template, and provides the editor for modifying templates.
Templates	List the base code templates. The active template is highlighted. The **Modified** field indicates whether you have changed the default template for the current language.
Stereotype Overrides	List the stereotyped templates, for the active base template. The **Modified** field indicates whether you have modified a default stereotyped template.
Add New Custom Template	Invoke a dialog for creating a custom stereotyped template.
Add New Stereotyped Override	Invoke a dialog for adding a stereotyped template, for the currently selected base template.

Option	Use to
Get Default Template	Update the editor display with the default version of the active template.
Save	Overwrite the active templates with the contents of the editor.
Delete	If you have overridden the active template, the override is deleted and replaced by the corresponding default code template.

For information on creating and editing code templates using the Code Template Editor window, see *SDK for Enterprise Architect*.

Note:

User-modified and user-defined Code Templates can be imported and exported as Reference Data (see the *Import and Export Reference Data* topic in *UML Model Management.* The templates defined for each language are indicated in the Export Reference Data dialog by the language name with the suffix *_Code_Templates*. If no templates exist for a language, there is no entry for the language in the dialog.

1.5.3 Synchronize Code

Enterprise Architect uses code templates during the forward synchronization of the following programming languages:

- ActionScript
- C
- C++
- C#
- Delphi
- Java
- PHP
- Python
- VB
- VB.Net

Only a subset of the code templates are used during synchronization. This subset corresponds to the distinct sections that Enterprise Architect recognizes in the source code. The following table lists the code templates and their corresponding code sections, which can be synchronized.

Code Template	Code Section
Class Notes	Comments preceding Class declaration.
Class Declaration	Up to and including Class parents.
Attribute Notes	Comments preceding Attribute declaration.
Attribute Declaration	Up to and including terminating character.
Operation Notes	Comments preceding operation declaration.
Operation Notes Impl	As for *Operation Notes*.
Operation Declaration	Up to and including terminating character.
Operation Declaration Impl	Up to and including terminating character.
Operation Body	Everything between and including the braces.
Operation Body Impl	As for *Operation Body*.

Three types of change can occur in the source when it is synchronized with the UML model:

- Synchronize Existing Sections 67: for example, changing the return type in an operation declaration
- Add New Sections to Existing Features 67: for example, adding notes to a Class declaration, where there were previously none
- Add New Features and Elements 67: for example, adding a new operation to a Class.

Each of these changes must be handled differently by Enterprise Architect; their effect on the CTF is described in the linked topics above.

1.5.3.1 Synchronize Existing Sections

When an existing section in the source code differs from the result generated by the corresponding template, that section is replaced. Consider for example, the following C++ Class declaration:

```
[asm] class A: public B
```

Now assume you add an inheritance relationship from Class A to Class C; the entire Class declaration would be replaced with something like:

```
[asm] class A: public B, public C
```

1.5.3.2 Add New Sections

The following can be added as new sections, to existing features in the source code:

- Class Notes
- Attribute Notes
- Operation Notes
- Operation Notes Impl
- Operation Body
- Operation Body Impl

Assume Class ***A*** from the previous example had no note when you originally generated the code. Now assume that you specify a note in the model for Class A. Enterprise Architect attempts to add the new note from the model during synchronization. It does this by executing the *Class Notes* template.

To make room for the new section to be inserted, you can specify how much white space to append to the section via synchronization macros. These macros are described in *SDK for Enterprise Architect*.

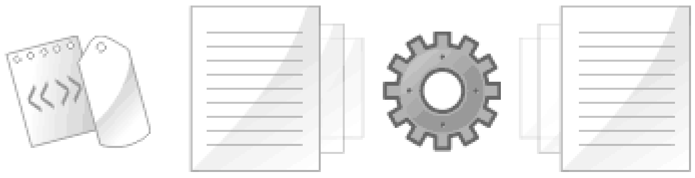
1.5.3.3 Add New Features and Elements

The following features and elements can be added to the source code during synchronization:

- Attributes
- Inner Classes
- Operations.

These are added by executing the relevant templates for each new element or feature in the model. Enterprise Architect attempts to preserve the appropriate indenting of new features in the code, by finding the indents specified in list macros of the Class. For languages that make use of namespaces, the *synchNamespaceBodyIndent* macro is available. Classes defined within a (non-global) namespace are indented according to the value set for this macro, during synchronization. This value is ignored for Classes defined within a package setup as a root namespace, or if the **Generate Namespace** option is set to **False** in the appropriate language page (C#, C++ or VB.Net) on the Options dialog (**Tools | Options | Source Code Engineering | <language>**).

1.6 Modeling Conventions



In order to get the most out of the round trip engineering in Enterprise Architect, you must be familiar with the modeling conventions used when generating and reverse engineering the languages you use. This topic describes the stereotypes, Tagged Values and other conventions used in code engineering in Enterprise Architect for the following supported languages:

- ActionScript 69
- Ada 2005 69 (for Systems Engineering and Ultimate editions of Enterprise Architect)
- C 71
- C# 73
- C++ 75
- Delphi 78
- Java 79
- PHP 80
- Python 81
- System C 81
- VB.Net 83
- Verilog 84
- VHDL 85
- Visual Basic 87

Note:

Enterprise Architect incorporates a number of visibility indicators or scope values for its supported languages. These include, for:

- All languages - Public (+), Protected (#) and Private (-)
- Java - Package (~)
- Delphi - Published (^)
- C# - Internal (~), Protected Internal (^)
- ActionScript - Internal (~)
- VB.NET - Friend (~), Protected Friend (^)
- PHP - Package (~)
- Python - Package (~)
- C - Package (~)
- C++ - Package (~).

1.6.1 ActionScript Conventions

Enterprise Architect supports round trip engineering of ActionScript 2 and 3, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
literal	Operation	A literal method referred to by a variable.
property get	Operation	A read property.
property set	Operation	A write property.

Tagged Values

Tag	Applies To	Corresponds To
attribute_name	Operation with stereotype *property get* or *property set*	The name of the variable behind this property.
dynamic	Class or Interface	The *dynamic* keyword.
final	ActionScript 3: Operation	The *final* keyword.
intrinsic	ActionScript 2: Class	The *intrinsic* keyword
namespace	ActionScript 3: Class, Interface, Attribute, Operation	The namespace of the current element.
override	ActionScript 3: Operation	The *override* keyword.
prototype	ActionScript 3: Attribute	The *prototype* keyword.
rest	ActionScript 3: Parameter	The *rest* parameter (...).

Common Conventions

- Package qualifiers (ActionScript 2) and Packages (ActionScript 3) are generated when the current package is not a namespace root 17
- An unspecified type is modeled as *var* or an empty **Type** field.

ActionScript 3 Conventions

- The *Is Leaf* property of a Class corresponds to the sealed keyword
- If a *namespace* tag is specified it overrides the *Scope* that is specified.

See Also

- Import Source Code 5
- Generate Source Code 12
- ActionScript Options 43

1.6.2 Ada 2005

Ada 2005 support is available in the System Engineering and Ultimate editions of Enterprise Architect.

Enterprise Architect supports round trip engineering of Ada 2005, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
adaPackage	Class	A package specification in Ada 2005 without a tagged record.
adaProcedure	Class	A procedure specification in Ada 2005.
delegate	Operation	Access to a subprogram.
enumeration	Inner Class	An *enum* type.
struct	Inner Class	A record definition.
typedef	Inner Class	A type definition, subtype definition, access type definition, renaming.

Tagged Values

Tag	Applies To	Corresponds To
Discriminant	Inner Class with stereotype *typedef*	The type's discriminant.
IsAccess	Parameter	Determination of whether the parameter is an access variable.
InstantiatedUnitType	Inner Class with stereotype *typedef*	The instantiated unit's type (*Package / Procedure / Function*).
PartType	Inner Class with stereotype *typedef*	The part type (*renames* or *new*).
Type	Inner Class with stereotype *typedef*	If Value = *SubType*, set subtype. If Value = *Access*, set access type.

Other Conventions

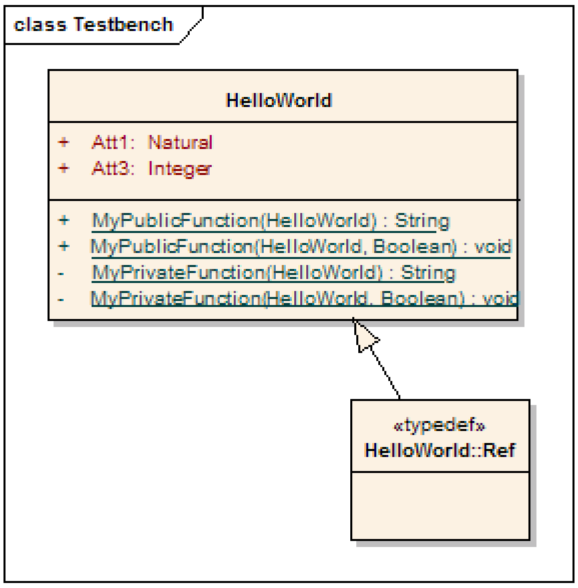
- Appropriate type of source files: Ada specification file, ***.ads***.
- Ada 2005 imports packages defined as either <<adaPackage>>Class or Class, based on the settings in the Ada options 43.
- A package in the Ada specification file is imported as a Class if it contains a Tagged Record, the name of which is governed by the options **Use Class Name for Tagged Record** and **Alternate Tagged Record Name**. All attributes defined in that Tagged Record are absorbed as the Class's attributes.
- A procedure / function in an Ada specification file is considered as the Class's member function if its first parameter satisfies the conditions specified in the options **Ref Param Style**, **Ignore Reference parameter name** and **Ref parameter name**.
- The option **Define Reference for Tagged Record**, if enabled, creates a reference type for the Class, the name of which is determined by the option **Reference Type Name**.

For example: *HelloWorld.ads*

```
package HelloWorld is
            type HelloWorld is tagged record
                    Att1: Natural;
                    Att3: Integer;
            end record;

            -- Public Functions
            function MyPublicFunction (P: HelloWorld) return String;
            procedure MyPublicFunction (P1: in out HelloWorld; AFlag: Boolean);
private
            -- Private Functions
            function MyPrivateFunction (P: HelloWorld) return String;
            procedure MyPrivateFunction (P1: in out HelloWorld; AFlag: Boolean);
```

end HelloWorld;


class Testbench
HelloWorld
+ Att1: Natural
+ Att3: Integer
+ MyPublicFunction(HelloWorld) : String
+ MyPublicFunction(HelloWorld, Boolean) : void
- MyPrivateFunction(HelloWorld) : String
- MyPrivateFunction(HelloWorld, Boolean) : void
«typedef»
HelloWorld::Ref

See Also

- Import Source Code 5
- Generate Source Code 12
- Ada 2005 Options 43

1.6.3 C Conventions

Note:

Separate conventions apply to Object Oriented programming in C 72.

Enterprise Architect supports round trip engineering of C, where the following conventions are used:

Stereotype

Stereotype	Applies To	Corresponds To
enumeration	Inner Class	An *enum* type.
struct	Inner Class	A *struct* type.
	Attribute	A keyword *struct* in variable definition.
typedef	Inner Class	A *typedef* statement, where the parent is the original type name.
union	Inner Class	A *union* type.
	Attribute	A keyword *union* in variable definition.

Tagged Values

Tag	Applies To	Corresponds To
anonymous	Class also containing the	The name of this class being defined only by

Tag	Applies To	Corresponds To
	Tagged Value *typedef*	the *typedef* statement.
bodyLocation	Operation	The location the method body is generated to. Expected values are **header**, **classDec** or **classBody.**
typedef	Class with stereotype other than *typedef*	This Class being defined in a *typedef* statement.

C Code Generation for UML Model

UML	C Code	Notes
A Class	A pair of C files (.h + .c)	File name is the same as Class name.
Operation (public & protected)	Function declaration in .h file and definition in .c file	
Operation (private)	Function definition in .c file only	
Attribute (public & protected)	Variable definition in .h file	
Attribute (private)	Variable definition in .c file	
Inner Class (without stereotype)	(N/A)	This inner Class would be ignored

See Also

- Import Source Code 5
- Generate Source Code 12
- C Options 44

1.6.3.1 *Object Oriented Programming In C*

The following conventions are used for Object-Oriented programming in C.

To configure Enterprise Architect to support Object-Oriented programming using C, you must set the **Object Oriented Support** option to **True** on the C Specifications 44 page of the Options dialog.

Stereotype

Stereotype	Applies To	Corresponds To
enumeration	Class	An *enum* type.
struct	Class	A *struct* type.
	Attribute	A keyword *struct* in variable definition.
typedef	Class	A *typedef* statement, where the parent is the original type name.
union	Class	A *union* type.
	Attribute	A keyword *union* in variable definition.

Tagged Values

Tag	Applies To	Corresponds To
anonymous	Class with stereotype of *enumeration*, *struct* or *union*.	The name of this Class being defined only by the *typedef* statement.
bodyLocation	Operation	The location the method body is generated to. Expected values are **header**, **classDec** or **classBody.**
define	Attribute	*#define* statement.
typedef	Class with stereotype of *enumeration*, *struct* or *union*.	This Class being defined in a *typedef* statement.

Object-Oriented C Code Generation for UML Model

The basic idea of implementing a UML Class in C code is to group the data variable (UML attributes) into a structure type. This structure is defined in a **.h** file so that it can be shared by other classes and by the client that referred to it.

An operation in a UML Class is implemented in C code as a function. The name of the function must be a fully qualified name that consists of the operation name, as well as the Class name to indicate that the operation is for that Class. A delimiter (specified in the **Namespace Delimiter** option on the C Specifications 44 page) is used to join the Class name and function (operation) name.

The function in C code must also have a reference parameter to the Class object. You can modify the **Reference as Operation Parameter**, **Reference Parameter Style** and **Reference Parameter Name** options on the C Specifications page to support this reference parameter.

Limitations of Object-Oriented Programming in C

1. No scope mapping for an attribute: an attribute in a UML Class is mapped to a structure variable in C code, and its scope (private, protected or public) is ignored.
2. Currently an inner Class is ignored: if a UML Class is the inner Class of another UML Class, it is ignored when generating C code.
3. Initial value is ignored: the initial value of an attribute in a UML Class is ignored in generated C code.

See Also

- Import Source Code 5
- Generate Source Code 12
- C Options 44

1.6.4 C# Conventions

Enterprise Architect supports the round trip engineering of C#, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
enumeration	Class	An *enum* type.
event	Operation	An event.
extension	Operation	A Class extension method, represented in code by a *this* parameter in the signature.
indexer	Operation	A property acting as an index for this Class.
partial	Operation	The *partial* keyword on an operation.

Stereotype	Applies To	Corresponds To
property	Operation	A property possibly containing both read and write code.
struct	Class	A *struct* type.

Tagged Values

Tag	Applies To	Corresponds To
argumentName	Operation with stereotype *extension*	The name given to the *this* parameter.
attributeName	Operation with stereotype *property* or *event*	The name of the variable behind this property or event.
className	Operation with stereotype *extension*	The Class that this method is being added to.
const	Attribute	The *const* keyword.
definition	Operation with stereotype *partial*	Whether this is the declaration of the method, or the definition.
delegate	Operation	The *delegate* keyword.
enumType	Operation with stereotype *property*	The datatype that the property is represented as.
extern	Operation	The *extern* keyword.
fixed	Attribute	The *fixed* keyword.
generic	Operation	The generic parameters for this Operation.
genericConstraints	Templated Class or Interface, Operation with tag *generic*	The constraints on the generic parameters of this type or operation.
Implements	Operation	The name of the method this implements, including the interface name.
ImplementsExplicit	Operation	The presence of the source interface name in this method declaration.
initializer	Operation	A constructor initialization list.
new	Class, Interface, Operation	The *new* keyword.
override	Operation	The *override* keyword.
params	Parameter	A parameter list using the params keyword.
partial	Class, Interface	The *partial* keyword.
readonly	Operation with stereotype *property*	This property only defining read code.
sealed	Operation	The *sealed* keyword.
static	Class	The *static* keyword.
unsafe	Class, Interface, Operation	The *unsafe* keyword.

Tag	Applies To	Corresponds To
virtual	Operation	The *virtual* keyword.
writeonly	Operation with stereotype *property*	This property only defining write code.

Other Conventions

- *Namespaces* are generated for each package below a namespace root 17
- The *Const* property of an attribute corresponds to the *readonly* keyword, while the tag *const* corresponds to the *const* keyword
- The value of *inout* for the *Kind* property of a parameter corresponds to the *ref* keyword
- The value of *out* for the *Kind* property of a parameter corresponds to the *out* keyword
- Partial Classes can be modeled as two separate Classes with the *partial* tag
- The *Is Leaf* property of a Class corresponds to the *sealed* keyword.

See Also

- Import Source Code 5
- Generate Source Code 12
- C# Options 45

1.6.5 C++ Conventions

Enterprise Architect supports round trip engineering of C++, including the Managed C++ 76 and C++/CLI 77 extensions, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
enumeration	Class	An *enum* type.
friend	Operation	The *friend* keyword.
property get	Operation	A read property.
property set	Operation	A write property.
struct	Class	A *struct* type.
typedef	Class	A *typedef* statement, where the parent is the original type name.
union	Class	A *union* type.

Tagged Values

Tag	Applies To	Corresponds To
afx_msg	Operation	The *afx_msg* keyword.
anonymous	Class also containing the Tagged Value *typedef*	The name of this class being only defined by the *typedef* statement.
attribute_name	Operation with stereotype *property get* or *property set*	The name of the variable behind this property.

Tag	Applies To	Corresponds To
bitfield	Attribute	The size, in bits, allowed for storage of this attribute.
bodyLocation	Operation	The location the method body is generated to; expected values are **header**, **classDec** or **classBody**.
callback	Operation	A reference to the **CALLBACK** macro.
explicit	Operation	The *explicit* keyword.
initializer	Operation	A constructor initialization list.
inline	Operation	The *inline* keyword and inline generation of the method body.
mutable	Attribute	The *mutable* keyword.
throws	Operation	The exceptions that are thrown by this method.
typedef	Class with stereotype other than *typedef*	This Class being defined in a *typedef* statement.
typeSynonyms	Class	The *typedef* name and/or fields of this type.
volatile	Operation	The *volatile* keyword.

Other conventions

- Namespaces are generated for each package below a namespace root 17
- *By Reference* attributes correspond to a pointer to the type specified
- The *Transient* property of an attribute corresponds to the *volatile* keyword
- The *Abstract* property of an attribute corresponds to the *virtual* keyword
- The *Const* property of an operation corresponds to the *const* keyword, specifying a constant return type
- The *Is Query* property of an operation corresponds to the *const* keyword, specifying the method doesn't modify any fields
- The *Pure* property of an operation corresponds to a *pure virtual* method using the "**= 0**" syntax
- The *Fixed* property of a parameter corresponds to the *const* keyword.

See Also

- Import Source Code 5
- Generate Source Code 12
- C++ Options 46

1.6.5.1 Managed C++ Conventions

The following conventions are used for managed extensions to C++ prior to C++/CLI 77. In order to set Enterprise Architect to generate managed C++ you must modify the C++ version in the C++ Options 46.

Stereotypes

Stereotype	Applies To	Corresponds To
property	Operation	The __*property* keyword.
property get	Operation	The __*property* keyword and a read property.
property set	Operation	The __*property* keyword and a write property.
reference	Class	The __*gc* keyword.

Stereotype	Applies To	Corresponds To
value	Class	The __*value* keyword.

Tagged Values

Tag	Applies To	Corresponds To
managedType	Class with stereotype *reference*, *value* or *enumeration;* Interface	The keyword used in declaration of this type. Expected values are *class* or *struct*.

Other Conventions

- The *typedef* and *anonymous* tags from native C++ are not supported
- The *Pure* property of an operation corresponds to the keyword __*abstract*.

See Also

- Import Source Code 5
- Generate Source Code 12

1.6.5.2 C++/CLI Conventions

The following conventions are used for modeling C++/CLI extensions to C++. In order to set Enterprise Architect to generate managed C++/CLI you must modify the C++ version in the C++ Options 46.

Stereotypes

Stereotype	Applies To	Description
event	Operation	Defines an event to provide access to the event handler for this Class.
property	Operation, Attribute	This is a property possibly containing both read and write code.
reference	Class	Corresponds to the *ref class* or *ref struct* keyword.
value	Class	Corresponds to the *value class* or *value struct* keyword.

Tagged Values

Tag	Applies To	Description
attribute_name	Operation with stereotype *property* or *event*	The name of the variable behind this property or event.
generic	Operation	Defines the generic parameters for this Operation.
genericConstraints	Templated Class or Interface, Operation with tag *generic*	Defines the constraints on the generic parameters for this Operation.
initonly	Attribute	Corresponds to the *initonly* keyword.
literal	Attribute	Corresponds to the *literal* keyword.
managedType	Class with stereotype *reference*, *value* or *enumeration;* Interface	Corresponds to either the *class* or *struct* keyword.

Other Conventions

- The *typedef* and *anonymous* tags are not used
- The *property get/property set* stereotypes are not used
- The *Pure* property of an operation corresponds to the keyword *abstract*.

See Also

- Import Source Code 5
- Generate Source Code 12

1.6.6 Delphi Conventions

Enterprise Architect supports round trip engineering of Delphi, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
constructor	Operation	A constructor.
destructor	Operation	A destructor.
dispinterface	Class, Interface	A dispatch interface.
enumeration	Class	An enumerated type.
metaclass	Class	A metaclass type.
object	Class	An object type.
operator	Operation	An operator.
property get	Operation	A read property.
property set	Operation	A write property.
struct	Class	A record type.

Tagged Values

Tag	Applies To	Corresponds To
attribute_name	Operation with stereotype *property get* or *property set*	The name of the variable behind this property.
overload	Operation	The *overload* keyword.
override	Operation	The *override* keyword.
packed	Class	The *packed* keyword.
property	Class	A property. See Delphi Properties 48 for more information.
reintroduce	Operation	The *reintroduce* keyword.

Other Conventions

- The *Static* property of an attribute or operation corresponds to the *class* keyword
- The *Fixed* property of a parameter corresponds to the *const* keyword
- The value of *inout* for the *Kind* property of a parameter corresponds to the *Var* keyword

- The value of *out* for the *Kind* property of a parameter corresponds to the *Out* keyword.

See Also

- Import Source Code 5
- Generate Source Code 12
- Delphi Options 47

1.6.7 Java Conventions

Enterprise Architect supports round trip engineering of Java - including AspectJ 80 extensions - where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
annotation	Interface	An *annotation* type.
enum	Attributes within a Class stereotyped *enumeration*	An *enumerated* option, distinguished from other attributes that have no stereotype.
enumeration	Class	An *enum* type.
operator	Operation	An operator.
property get	Operation	A read property.
property set	Operation	A write property.
static	Class or Interface	The *static* keyword.

Tagged Values

Tag	Applies To	Corresponds To
annotations	Anything	The annotations on the current code feature.
arguments	Attribute with stereotype *enum*	The arguments that apply to this enumerated value.
attribute_name	Operation with stereotype *property get* or *property set*	The name of the variable behind this property.
dynamic	Class or Interface	The *dynamic* keyword.
generic	Operation	The generic parameters to this operation.
parameterList	Parameter	A parameter list with the **...** syntax.
throws	Operation	The exceptions that are thrown by this method.
transient	Attribute	The transient keyword.

Other Conventions

- Package statements are generated when the current package is not a namespace root 17
- The *Const* property of an attribute or operation corresponds to the final keyword
- The *Transient* property of an attribute corresponds to the volatile keyword
- The *Fixed* property of a parameter corresponds to the final keyword.

See Also

- Import Source Code 5
- Generate Source Code 12
- Java Options 51

1.6.7.1 AspectJ Conventions

The following are the conventions used for supporting AspectJ extensions to Java.

Stereotypes

Stereotype	Applies To	Corresponds To
advice	Operation	A piece of advice in an AspectJ aspect.
aspect	Class	An AspectJ aspect.
pointcut	Operation	A pointcut in an AspectJ aspect.

Tagged Values

Tag	Applies To	Corresponds To
className	Attribute or operation within a Class stereotyped *aspect*	The Classes this AspectJ intertype member belongs to.

Other Conventions

- The specifications of a pointcut are included in the **Behavior** field of the method.

See Also

- Import Source Code 5
- Generate Source Code 12

1.6.8 PHP Conventions

Enterprise Architect supports the round trip engineering of PHP 4 and 5, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
property get	Operation	A read property.
property set	Operation	A write property.

Tagged Values

Tag	Applies To	Corresponds To
attribute_name	Operation with stereotype *property get* or *property set*	The name of the variable behind this property.
final	Operations in PHP 5.	The final keyword.

Common Conventions

- An unspecified type is modeled as *var*

- Methods returning a reference are generated by setting the *Return Type* to *var**
- Reference parameters are generated from parameters with the parameter *Kind* set to *inout* or *out*.

PHP 5 Conventions

- The *final* Class modifier corresponds to the *Is Leaf* property
- The *abstract* Class modifier corresponds to the *Abstract* property
- Parameter type hinting is supported by setting the *Type* of a parameter
- The value of inout or *out* for the Kind property of a parameter corresponds to a *reference* parameter.

See Also

- Import Source Code 5
- Generate Source Code 12
- PHP Options 51

1.6.9 Python Conventions

Enterprise Architect supports the round trip engineering of Python, where the following conventions are used.

Tagged values

Tag	Applies To	Corresponds To
decorators	Class, Operation	The decorators applied to this element in the source.

Other Conventions

- Model members with *Private Scope* correspond to code members with two leading underscores
- Attributes are only generated when the Initial value is not empty
- All types are reverse engineered as *var*.

See Also

- Import Source Code 5
- Generate Source Code 12
- Python Options 52

1.6.10 System C Conventions

Enterprise Architect supports round-trip engineering of SystemC, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
delegate	Method	A delegate.
enumeration	Inner Class	An *enum* type.
friend	Method	A *friend* method.
property	Method	A property definition.
sc_ctor	Method	A SystemC constructor.
sc_module	Class	A SystemC module.
sc_port	Attribute	A port.
sc_signal	Attribute	A signal

Stereotype	Applies To	Corresponds To
struct	Inner Class	A *struct* or *union*.

Tagged Values

Tag	Applies To	Corresponds To
kind	Attribute (Port)	Port kind (*clocked*, *fifo*, *master*, *slave*, *resolved*, *vector*).
mode	Attribute (Port)	Port mode (*in*, *out*, *inout*).
overrides	Method	The *Inheritance* list of a method declaration.
throw	Method	The exception specification of a method.

Other Conventions

- SystemC also inherits most of the stereotypes and Tagged Values of C++ 75.

SystemC Toolbox Pages

To access the SystemC pages of the Enterprise Architect UML Toolbox, select the **More tools | HDL | SystemC Constructs** menu option. Drag these icons onto a diagram to model a SystemC design.

Page	Item	Use To
SystemC	Module	Define a SystemC Module. An *sc_module*-stereotyped Class element.
	Enumeration	Define an Enumerated Type. An *enumeration*-stereotyped Enumeration element.
	Struct	Define a Structure. A *struct*-stereotyped Class element.
SystemC Features	Port	Define a SystemC Port. An *sc_port*-stereotyped attribute.
	Signal	Define a SystemC Signal. An *sc_signal*-stereotyped attribute.
	Constructor	Define a SystemC Constructor. An *sc_ctor*-stereotyped method.

See Also

- Import Source Code 5
- Generate Source Code 12
- SystemC Language Options 53.

1.6.11 VB.Net Conventions

Enterprise Architect supports round-trip engineering of Visual Basic.Net, where the following conventions are used. Earlier versions of Visual Basic 87 are supported as a different language.

Stereotypes

Stereotype	Applies To	Corresponds To
event	Operation	An event declaration.
import	Operation	An operation to be imported from another library.
module	Class	A module.
operator	Operation	An operator overload definition.
partial	Operation	The *partial* keyword on an operation.
property	Operation	A property possibly containing both read and write code.

Tagged Values

Tag	Applies To	Corresponds To
Alias	Operation with stereotype *import*	The alias for this imported operation.
attribute_name	Operation with stereotype *property*	The name of the variable behind this property.
Charset	Operation with stereotype *import*	The *character set* clause for this import. One of the values *Ansi, Unicode* or *Auto*.
delegate	Operation	The *Delegate* keyword.
enumTag	Operation with stereotype *property*	The datatype that this property is represented as.
Handles	Operation	The *handles* clause on this operation.
Implements	Operation	The *implements* clause on this operation.
Lib	Operation with stereotype *import*	The library this import comes from.
MustOverride	Operation	The *MustOverride* keyword.
Narrowing	Operation with stereotype *operator*	The *Narrowing* keyword.
NotOverrideable	Operation	The *NotOverrideable* keyword.
Overloads	Operation	The *Overloads* keyword.
Overrides	Operation	The *Overrides* keyword.
parameterArray	Parameter	A parameter list using the *ParamArray* keyword.
partial	Class, Interface	The *Partial* keyword.
readonly	Operation with stereotype *property*	This property only defining read code.
shadows	Class, Interface, Operation	The *Shadows* keyword.
Shared	Attribute	The *Shared* keyword.

Tag	Applies To	Corresponds To
Widening	Operation with stereotype *operator*	The *Widening* keyword.
writeonly	Operation with stereotype *property*	This property only defining write code.

Other Conventions

- Namespaces are generated for each package below a namespace root 17
- The *Is Leaf* property of a Class corresponds to the *NotInheritable* keyword
- The *Abstract* property of a Class corresponds to the *MustInherit* keyword
- The *Static* property of an attribute or operation corresponds to the *Shared* keyword
- The *Abstract* property of an operation corresponds to the *MustOverride* keyword
- The value of *in* for the *Kind* property of a parameter corresponds to the *ByVal* keyword
- The value of *inout* or *out* for the *Kind* property of a parameter corresponds to the *ByRef* keyword.

See Also

- Import Source Code 5
- Generate Source Code 12
- VB.Net Options 54

1.6.12 Verilog Conventions

Enterprise Architect supports round-trip engineering of Verilog, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
asynchronous	Method	A concurrent process.
enumeration	Inner Class	An *enum* type.
initializer	Method	An initializer process.
module	Class	A module.
part	Attribute	A component instantiation.
port	Attribute	A port.
synchronous	Method	A sequential process.

Tagged Values

Tag	Applies To	Corresponds To
kind	Attribute (signal)	The signal kind (such as *register*, *bus*).
mode	Attribute (port)	The port mode (*in*, *out*, *inout*).
Portmap	Attribute (part)	The generic / port map of the component instantiated.
sensitivity	Method	The sensitivity list of a sequential process.
type	Attribute	The range or type value of an attribute.

Verilog Toolbox Pages

To access the Verilog pages of the Enterprise Architect UML Toolbox, select the **More tools | HDL | Verilog Constructs** menu option. Drag these icons onto a diagram to model a Verilog design.

Page	Item	Use To
Verilog	Module	Define a Verilog Module. A *module*-stereotyped Class element.
	Enumeration	Define an Enumerated Type. An *enumeration*-stereotyped Class element.
Verilog Features	Port	Define a Verilog Port. A *port*-stereotyped attribute.
	Part	Define a Verilog component instantiation A *part*-stereotyped attribute.
	Attribute	Define an attribute.
	Procedure • Concurrent • Sequential • Initializer.	Define a Verilog process: • An *asynchronous*-stereotyped method • A *synchronous*-stereotyped method • An *initializer*-stereotyped method.

See Also

- Import Source Code 5
- Generate Source Code 12
- Verilog Language Options. 55

1.6.13 VHDL Conventions

Enterprise Architect supports round-trip engineering of VHDL, where the following conventions are used.

Stereotypes

Stereotype	Applies To	Corresponds To
architecture	Class	An architecture.
asynchronous	Method	An asynchronous process.
configuration	Method	A configuration.
enumeration	Inner Class	An *enum* type.
entity	Interface	An entity.
part	Attribute	A component instantiation.
port	Attribute	A port.
signal	Attribute	A signal declaration.
struct	Inner Class	A record definition.
synchronous	Method	A synchronous process.

Stereotype	Applies To	Corresponds To
typedef	Inner Class	A *type* or *subtype* definition.

Tagged Values

Tag	Applies To	Corresponds To
isGeneric	Attribute (port)	The port declaration in a generic interface.
isSubType	Inner Class (typedef)	A subtype definition.
kind	Attribute (signal)	The signal kind (such as *register*, *bus*).
mode	Attribute (port)	The port mode (*in*, *out*, *inout, buffer, linkage*).
portmap	Attribute (part)	The generic / port map of the component instantiated.
sensitivity	Method (synchronous)	The sensitivity list of a synchronous process.
type	Inner Class (typedef)	The type indication of a type declaration.
typeNameSpace	Attribute (part)	The type namespace of the instantiated component.

VHDL Toolbox Pages

To access the VHDL pages of the Enterprise Architect UML Toolbox, select the **More tools | HDL | VHDL Constructs** menu option. Drag these icons onto a diagram to model a VHDL design.

Page	Item	Use To
VHDL	Architecture	Define an architecture to be associated with a VHDL entity. An *architecture*-stereotyped Class element.
	Entity	Define a VHDL entity to contain the Port definitions. An *entity*-stereotyped interface element.
	Enumeration	Define an Enumerated Type. An *enumeration*-stereotyped enumeration element.
	Struct	Define a VHDL record. A *struct*-stereotyped Class element.
	Typedef	Define a VHDL type or subtype A *typedef*-stereotyped Class element.
VHDL Features	Port	Define a VHDL Port. A *port*-stereotyped attribute.
	Part	Define a VHDL component instantiation A *part*-stereotyped attribute.
	Signal	Define a VHDL signal. A *signal*-stereotyped attribute.
	Procedure	Define a VHDL process:

Page	Item	Use To
	• Concurrent • Sequential • Configuration .	• An *asynchronous*-stereotyped method • A *synchronous*-stereotyped method • A *configuration*-stereotyped method.

See Also

- Import Source Code 5
- Generate Source Code 12
- VHDL Language Options. 56

1.6.14 Visual Basic Conventions

Enterprise Architect supports the round trip engineering of Visual Basic 5 and 6, where the following conventions are used. Visual Basic .Net 83 is supported as a different language.

Stereotypes

Stereotype	Applies To	Corresponds To
global	Attribute	The *Global* keyword.
import	Operation	An operation to be imported from another library.
property get	Operation	A property get.
property set	Operation	A property set.
property let	Operation	A property let.
with events	Attribute	The *WithEvents* keyword.

Tagged Values

Tag	Applies To	Corresponds To
Alias	Operation with stereotype *import*	The alias for this imported operation.
attribute_name	Operation with stereotype *property get* , *property set* or *property let*	The name of the variable behind this property.
Lib	Operation with stereotype *import*	The library this import comes from.
New	Attribute	The *New* keyword.

Other Conventions

- The value of *in* for the *Kind* property of a parameter corresponds to the *ByVal* keyword
- The value of *inout* or *out* for the *Kind* property of a parameter corresponds to the *ByRef* keyword.

See Also

- Import Source Code 5
- Generate Source Code 12
- Visual Basic Options 57

2 XML Technologies

Enterprise Architect enables rapid modeling, forward engineering and reverse engineering of two key W3C XML technologies:

- XML Schema (XSD)
- Web Service Definition Language (WSDL).

XSD and WSDL support is critical for the development of a complete *Service Oriented Architecture* (SOA), and the coupling of UML 2.1 and XML provides the natural mechanism for specifying, constructing and deploying XML-based SOA artifacts within an organization.

The following topics explain how to work with these technologies using Enterprise Architect.

2.1 XML Schema (XSD)

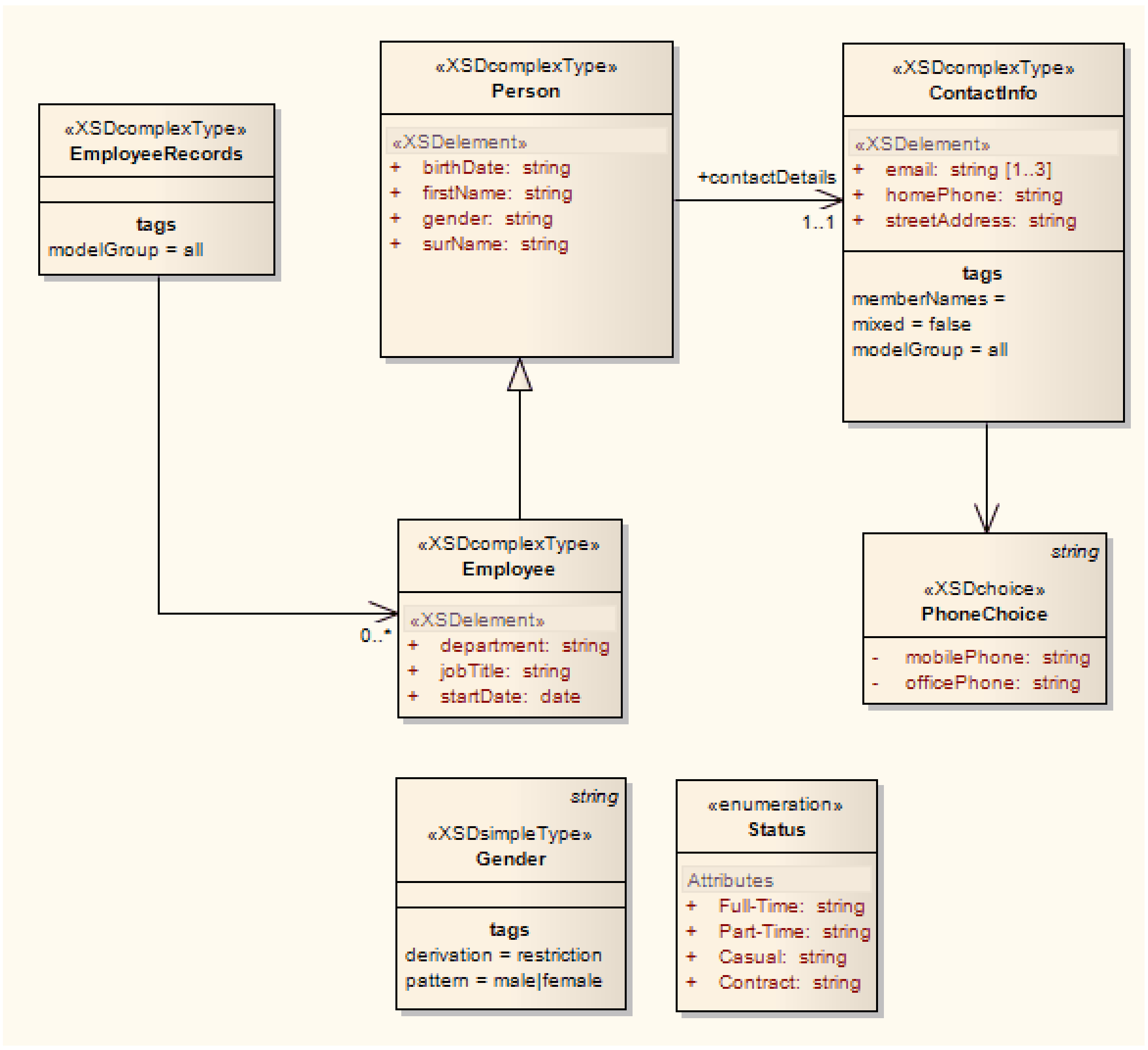
Enterprise Architect supports Forward and Reverse engineering of W3C XML schemas (XSD). The following topics explain how to use Enterprise Architect to model, generate and import XML schemas:

- *Model XSD* 89
- *Import XSD* 102
- *Generate XSD* 100

2.1.1 Model XSD

XML schemas are modeled using UML Class diagrams. The XML Schema pages of the Enterprise Architect UML Toolbox (see *Using Enterprise Architect - UML Modeling Tool*) provide in-built support for the UML profile for XSD. This enables an abstract UML Class model to be automatically generated as a W3C XML Schema (XSD) file.

The following Class diagram models simple schema for an example *Employee Details* system, intended to store a company's employee contact information. The Classes shown form the *EmployeeDetails* package. The UML attributes of the Classes map directly to XML elements or attributes. Note that the Classes have no methods, since there is no meaningful correspondence between Class methods and XSD constructs.



The following code shows the schema generated for the *Employee Details* package by default. Notice how each UML Class corresponds to a *complexType* definition in the schema. The Class attributes are generated as schema elements contained in a Sequence model group within the definition. The *Enumeration* Class is the exception here - it maps directly to an XSD enumeration, contained within a *simpleType* definition.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
	<xs:element name="ContactInfo" type="ContactInfo"/>
	<xs:complexType name="ContactInfo">
		<xs:sequence>
			<xs:element name="ContactInfo.homePhone" type="xs:string" maxOccurs="1"/>
			<xs:element name="ContactInfo.email" type="xs:string"/>
			<xs:element name="ContactInfo.streetAddress" type="xs:string"/>
			<xs:choice>
				<xs:element name="ContactInfo.mobilePhone" type="xs:string"/>
				<xs:element name="ContactInfo.officePhone" type="xs:string"/>
			</xs:choice>
		</xs:sequence>
	</xs:complexType>
	<xs:simpleType name="Gender">
		<xs:restriction base="xs:string">
			<xs:pattern value="male|female"/>
		</xs:restriction>
	</xs:simpleType>
	<xs:element name="Employee" type="Employee"/>
	<xs:complexType name="Employee">
		<xs:complexContent>
```

```
                        <xs:extension base="Person">
                                <xs:sequence>
                                        <xs:element name="status" type="Status"/>
                                        <xs:element name="jobTitle" type="xs:string"/>
                                        <xs:element name="startDate" type="xs:date"/>
                                        <xs:element name="department" type="xs:string"/>
                                </xs:sequence>
                        </xs:extension>
                </xs:complexContent>
        </xs:complexType>
        <xs:element name="Person" type="Person"/>
        <xs:complexType name="Person">
                <xs:sequence>
                        <xs:element name="surName" type="xs:string" maxOccurs="1"/>
                        <xs:element name="firstName" type="xs:string" maxOccurs="1"/>
                        <xs:element name="birthDate" type="xs:string" maxOccurs="1"/>
                        <xs:element name="contactDetails" type="ContactInfo"/>
                </xs:sequence>
                <xs:attribute name="gender" use="optional" type="Gender"/>
        </xs:complexType>
        <xs:element name="EmployeeRecords" type="EmployeeRecords"/>
        <xs:complexType name="EmployeeRecords">
                <xs:all>
                        <xs:element name="Employee" type="Employee"/>
                </xs:all>
        </xs:complexType>
        <xs:simpleType name="Status">
                <xs:restriction base="xs:string">
                        <xs:enumeration value="Full-Time"/>
                        <xs:enumeration value="Part-Time"/>
                        <xs:enumeration value="Casual"/>
`                       <xs:enumeration value="Contract"/>
                </xs:restriction>
        </xs:simpleType>
</xs:schema>
```

The following topics provide further explanation:

- *UML Profile for XSD* 91
- *XSD Datatypes Package* 97
- *Abstract XSD Models* 98

2.1.1.1 UML Profile for XSD

The UML Profile for XSD specifies a set of stereotypes, Tagged Values and constraints that can be applied to the UML model in order to change particular aspects of the resulting schema. For example, you might have to convert certain UML Class attributes to XSD attributes, or use a model group other than the default *Sequence*.

Enterprise Architect provides native support for the UML Profile for XSD via the XML schema pages of the Enterprise Architect UML Toolbox (see *Using Enterprise Architect - UML Modeling Tool*). Alternatively, you can use the profile via Enterprise Architect's generic profile mechanism by downloading the UML Profile for XSD. See the *Using Profiles* topic in *Extending UML With Enterprise Architect* for details on importing UML profiles into Enterprise Architect. The XSD profile used by Enterprise Architect is an adaptation of the profile defined in *Modeling XML Applications with UML* (David Carlson).

The XSD stereotypes provide an explicit mapping from XSD to UML constructs. The Tagged Values further define aspects of the mapping, such as whether the elements should be qualified. Full information on the Tagged Values can be obtained from the W3C XML Schema recommendation. The constraints define any conditions that must be satisfied for the stereotype to apply.

The following stereotypes are provided:

- XSDschema 92
- XSDcomplexType 93
- XSDsimpleType 93
- XSDsequence 94
- XSDchoice 94
- XSDelement 94
- XSDattribute 95

- XSDany 95
- XSDrestriction 95
- XSDgroup 96
- XSDtopLevelElement 96
- XSDtopLevelAttribute 96
- XSDunion 97
- XSDattributeGroup 97.

The following tables list the features of the UML Profile for XSD.

Note:

Tagged Value names are shown in bold followed by the allowed values.

If a default value is used by Enterprise Architect's schema generator, it is underlined.

«XSDschema»

UML Construct		Package
Description		All Classes in a package are defined within one schema. This stereotype can be used to specify schema-wide settings.
Tagged Values	**anonymousRole:** (true | <u>false</u>)	Specifies if the role name is included in the element declaration for the UML attribute.
	anonymousType: (true | <u>false</u>)	Specifies whether the Class type is anonymous for attributes.
	attributeFormDefault: (qualified | <u>unqualified</u>)	Determines whether attribute instances must be qualified.
	defaultNamespace:	The default namespace used in this schema. This value is used to specify the default namespace attribute (*xmlns*=), in the schema element.
	elementDerivation: (<u>true</u> | false)	Determines whether inheritances are generated using XSD extension or copy-down inheritance.
	elementFormDefault: (qualified | <u>unqualified</u>)	Determines whether element instances must be qualified.
	memberNames: (<u>qualified</u> | unqualified)	Determines whether elements generated from Class attributes have their name qualified by the corresponding Class name.
	modelGroup: (all | <u>sequence</u> | choice)	Specifies the default XSD model group used to generate *complexType* definitions.
	schemaLocation:	The URI that identifies the location of the schema. This value is used in the import and include elements.
	targetNamespace:	The URI that uniquely identifies this schema's namespace.
	targetNamespacePrefix:	The prefix that abbreviates the *targetNamespace*.
	version:	The version of this schema.
Constraints		None.

«XSDcomplexType»

UML Construct		Class
Description		*complexType* definitions are created for generic UML Classes. This stereotypes helps tailor the generation of a *complexType* definition.
Tagged Values	**memberNames:** (qualified | unqualified)	Determines whether elements generated from the UML Class attributes and associations have their name qualified by the corresponding Class name for this *complexType* definition.
	mixed: (true | false)	Determines whether this element can contain mixed element and character content. See the W3CXML Schema recommendation.
	modelGroup: (all | sequence | choice)	Overrides the default XSD model for generating this *complexType* definition.
Constraints		None.

«XSDsimpleType»

UML Construct		Class
Description		An XSD *simpleType* is generated for Classes with this stereotype.
Tagged Values	**derivation:** (restriction | list)	Specifies the derivation of the *simpleType*. See the W3C XML Schema recommendation.
	length:	See the W3C XML Schema recommendation.
	minLength:	
	maxLength:	
	minInclusive:	
	minExclusive:	
	maxInclusive:	
	maxExclusive:	
	totalDigits:	
	fractionDigits:	
	whiteSpace:	
	pattern:	
Constraints		This Class can only participate in an inheritance relation with another *simpleType*. It cannot have any attributes or own any associations; they are ignored if present.

«XSDsequence»

UML Construct		Class
Description		The schema generator creates a sequence model group as the container for the attributes and associations owned by this Class. The model group is in turn added to the model groups of this Class respective owners. **Note:** Tagged values specified by owners of this Class persist through to the child elements of this model group. Thus if memberNames are unqualified for a complexType, so are the children of this model group when added to that complexType.
Tagged Values		None.
Constraints		This Class must be the destination of unidirectional associations. If it is not, this Class and its connectors are ignored, possibly invalidating other model group Classes. Inheritance relations are ignored for this Class.

«XSDchoice»

UML Construct		Class
Description		Creates an XSD choice element. See *XSDsequence* for more details.
Tagged Values		None.
Constraints		As for *XSDsequence*.

«XSDelement»

UML Construct		Attribute: *AssociationEnd*
Description		By applying this stereotype to a UML Class attribute or *AssociationEnd*, the corresponding UML entity is generated as an element within the parent *complexType* and not as an XSD attribute.
Tagged Values	**form:** (qualified | unqualified)	Overrides the schema's *elementFormDefault* value.
	position:	Causes the elements to be ordered within a sequence model group of the containing *complexType*. Duplicated and invalid position Tagged Values are ignored and result in undefined ordering of the UML attributes. Missing position values cause the defined positions to be allocated as specified, with the remaining elements filling the missing positions in an undefined order.
	anonymousRole: (true | false)	Specifies if the role name is included in the element declaration for the UML attribute.

	anonymousType: (true | false)	Specifies whether the Class type is anonymous for attributes.
	default	See the W3C XML Schema recommendation.
	fixed	
Constraints		None.

«XSDattribute»

UML Construct		Attribute: *AssociationEnd*
Description		By applying this stereotype to a UML Class attribute or *AssociationEnd*, the corresponding UML entity is generated as an XSD attribute within the parent *complexType* and not as an XSD element.
Tagged Values	**form:** (qualified | unqualified)	Overrides the schema's *attributeFormDefault* value.
	use: (prohibited | optional | required)	See the W3C XML Schema recommendation.
	default	
	fixed	
Constraints		The attribute *datatype* should not see a Class specification, otherwise it is ignored.

«XSDany»

UML Construct		Class: *Attribute*
Description		If applied to a UML attribute, an XSD *anyAttribute* element is generated. If applied to a UML Class, an XSD *any* element is generated.
Tagged Values	**namespace:**	See the W3C XML Schema recommendation.
	processContents: (skip | lax | strict)	
Constraints		None.

«XSDrestriction»

UML Construct		Generalization
Description		Overrides the default use of XSD extension for inheritance and generates the child as a *complexType* with a restriction element instead.
Tagged		None.

Values		
Constraints		Applies only to UML Class parent-child relations.

«XSDgroup»

UML Construct		Class
Description		An *XSDgroup* is generated for Classes with this stereotype.
Tagged Values	**modelGroup:** (sequence | choice | all)	Overrides the default XSD model for generating this group definition.
Constraints		A group Class can only associate itself to other group Classes. A group Class can be associated by another group Class or a *complexType* Class. The association should be via an Association connector. A group Class cannot be inherited/aggregated.

«XSDtopLevelElement»

UML Construct		Class
Description		Creates an <*xs:element*> construct which acts as a container for *XSDcomplexType* and *XSDsimpleType* Class.
Tagged Values	**default**	See the W3C XML Schema recommendation.
	fixed	
Constraints		An *XSDtopLevelElement* Class can contain either an *XSDsimpleType* or an *XSDcomplexType* as its child Class. When such a Class is present as its child, all its inheritance is ignored. This Class cannot be inherited.

«XSDtopLevelAttribute»

UML Construct		Class
Description		Creates an <*xs:attributr*> construct which acts as a container for *XSDsimpleType* Class.
Tagged Values	**use:** (optional | required | prohibited)	See the W3C XML Schema recommendation.
	default	
	fixed	
Constraints		An *XSDtopLevelAttribute* Class can contain only an *XSDsimpleType* Class as its child Class. When such a Class is present as its child, all its inheritance is ignored.

		This Class can inherit from only one *XSDsimpleType* Class.

«XSDunion»

UML Construct		Class
Description		Creates an <*xs:union*> construct which can act as a container for *XSDsimpleType* Class.
Tagged Values		None
Constraints		An *XSDunion* Class can contain only *XSDsimpleType* as its child Class and can generalize from other *XSDsimpleType* Classes only. All the Classes that this Class generalizes become the members of the attribute *memberTypes*. This Class cannot have any attributes or associations.

«XSDattributeGroup»

UML Construct		Class
Description		Creates an <*XSDattributeGroup*> construct which can act as a container for a set of elements for stereotype *XSDattribute*.
Tagged Values		None
Constraints		An *XSDattributeGroup* Class can contain only elements of stereotype *XSDattribute* and can be associated only with other *XSDattributeGroup* Classes. Only *XSDcomplexType* Classes can associate with this Class. This Class cannot be inherited.

2.1.1.2 XSD Datatypes Package

When modeling XSD constructs, it is often useful to have the XSD primitive types represented as UML elements. In this way user-defined types, for example, can reference the datatype elements as part of inheritance or association relationships.

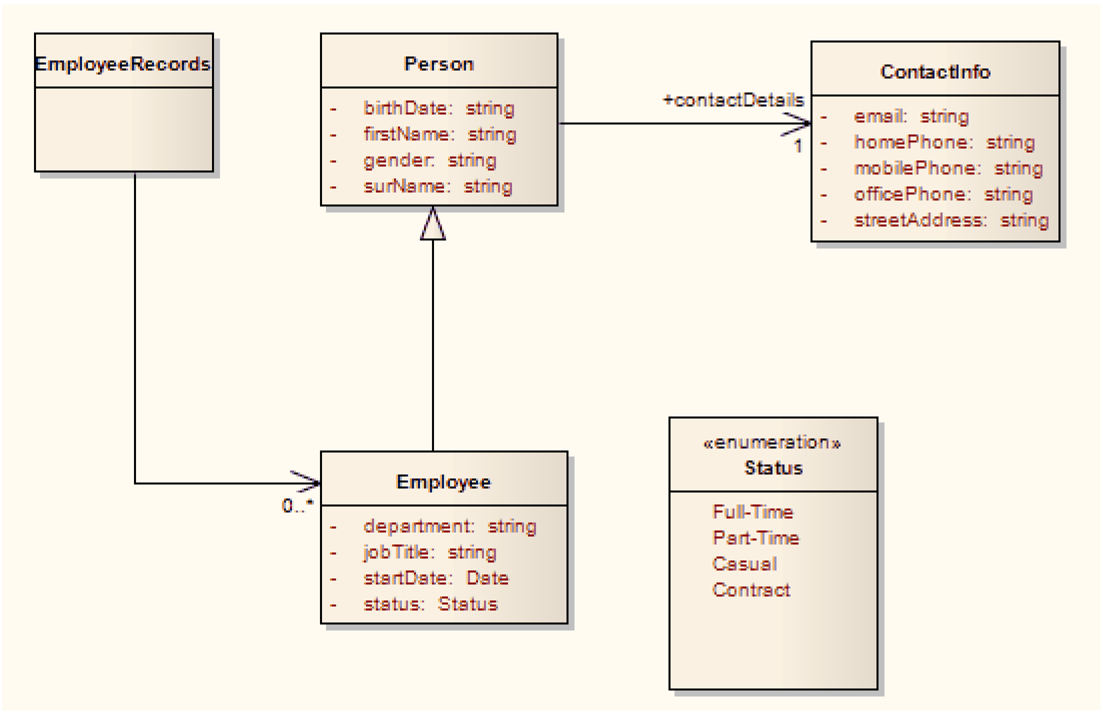
Sparx Systems provides the set of primitive XSD data types as a UML package in the form of an XMI file. Each of the XSD primitive types is represented by a UML Class in a package named *XSDDatatypes*. To import the *XSDDatatypes* package into your model, follow the steps below:

1. Download the *XSDDatatypes* package using the following link: XSDDatatypes Package. The file *XSDDataTypes.xml* is an XMI file.
2. Use Enterprise Architect's XMI import facility, which is available via the **Project | Import/Export | Import Package from XMI** menu option. See the *Import XMI* topic in *UML Model Management.*
3. When the XMI import is complete, you have the UML package named *XSDDatatypes* in your model, from which you can drag and drop the relevant types as required.

2.1.1.3 Abstract XSD models

XML schemas can be modeled using simple, abstract Class models. This can be useful in enabling an architect to start work at a higher level of abstraction, without concern for the implementation details of a schema. Such an abstract model can be refined further using the XML Schema pages of the Enterprise Architect UML Toolbox (see *Using Enterprise Architect - UML Modeling Tool*), or it can be generated directly by Enterprise Architect's schema generator 100. In this case, a set of default mappings 99 is assumed by the schema generator to convert the abstract model to an XSD file.

The following is a simplified version of the Employee Details example model, which does not use XSD-specific stereotypes or Tagged Values.



The following schema fragment would be generated by Enterprise Architect, given the above model.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="Status">
            <xs:restriction base="xs:string">
                    <xs:enumeration value="Full-Time"/>
                    <xs:enumeration value="Part-Time"/>
                    <xs:enumeration value="Casual"/>
                    <xs:enumeration value="Contract"/>
            </xs:restriction>
    </xs:simpleType>
    <xs:element name="Person" type="Person"/>
    <xs:complexType name="Person">
            <xs:sequence>
                    <xs:element name="firstName" type="xs:string"/>
                    <xs:element name="surName" type="xs:string"/>
                    <xs:element name="birthDate" type="xs:string"/>
                    <xs:element name="gender" type="xs:string"/>
                    <xs:element name="contactDetails" type="ContactInfo"/>
            </xs:sequence>
    </xs:complexType>
    <xs:element name="Employee" type="Employee"/>
    <xs:complexType name="Employee">
            <xs:complexContent>
                    <xs:extension base="Person">
                            <xs:sequence>
                                    <xs:element name="status" type="Status"/>
                                    <xs:element name="jobTitle" type="xs:string"/>
                                    <xs:element name="startDate" type="xs:date"/>
                                    <xs:element name="department" type="xs:string"/>
```

```
                        </xs:sequence>
                    </xs:extension>
                </xs:complexContent>
            </xs:complexType>
            <xs:element name="EmployeeRecords" type="EmployeeRecords"/>
            <xs:complexType name="EmployeeRecords">
                <xs:sequence>
                    <xs:element name="Employee" type="Employee" minOccurs="0" maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:complexType>
            <xs:element name="ContactInfo" type="ContactInfo"/>
            <xs:complexType name="ContactInfo">
                <xs:sequence>
                    <xs:element name="homePhone" type="xs:string"/>
                    <xs:element name="mobilePhone" type="xs:string"/>
                    <xs:element name="officePhone" type="xs:string"/>
                    <xs:element name="email" type="xs:string"/>
                    <xs:element name="streetAddress" type="xs:string"/>
                </xs:sequence>
            </xs:complexType>
        </xs:schema>
```

2.1.1.3.1 Default UML to XSD Mappings

The following table describes the default mapping of UML to XSD constructs. This set of mappings is useful when defining simple schemas from abstract Class models. The defaults are also assumed by the schema generator when generating unstereotyped elements in an abstract model. The XML Schema pages of the Enterprise Architect UML Toolbox (and UML Profile for XSD) override these default mappings through the use of stereotypes and Tagged Values.

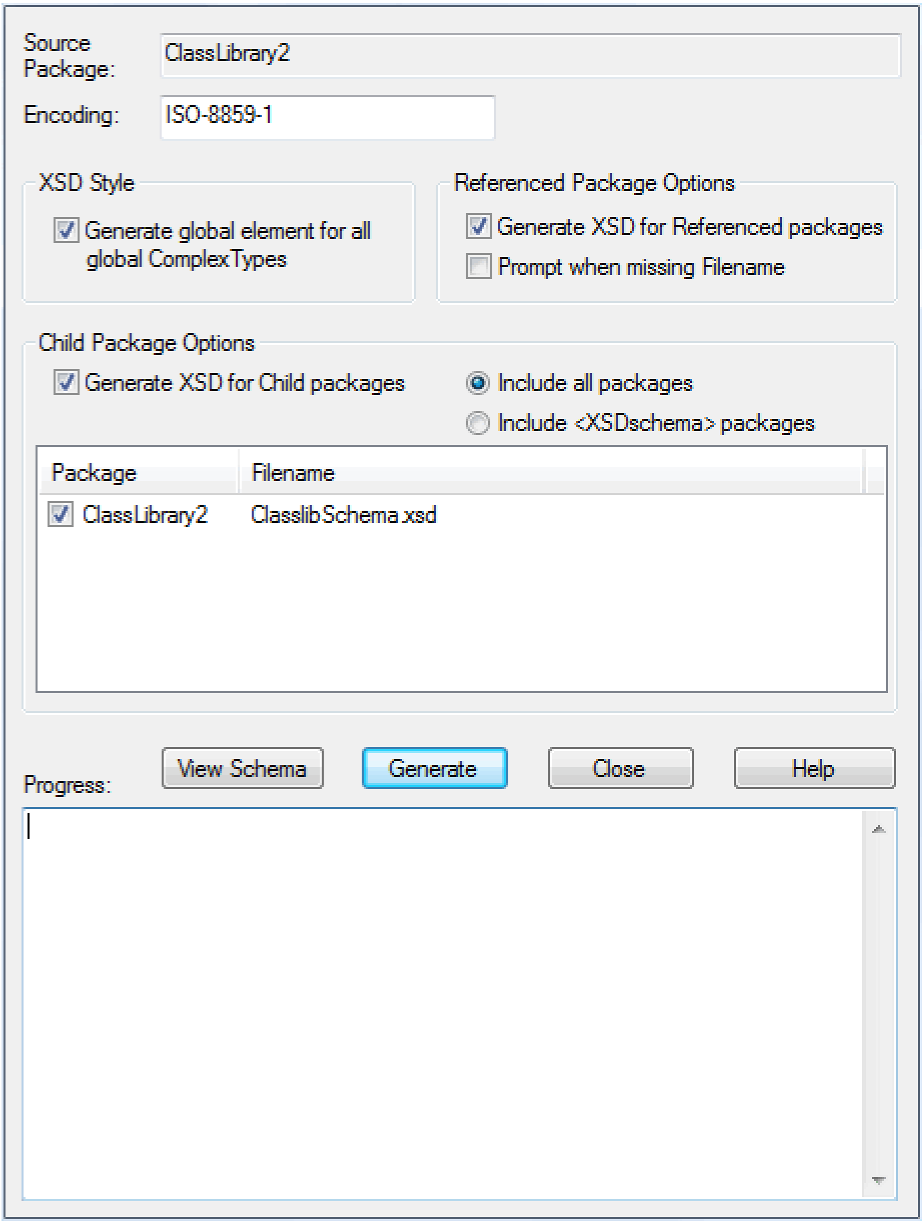
UML Construct	Default XSD Production Rules
Package	A schema element is generated for the target package. If the target package includes Classes from another package, which has the Tagged Values *targetNamespace* and *targetNamespacePrefix* set, these are included as attributes of the schema element. In addition, an *import* or *include* element is created for each referenced package. (An *include* element is used if the external package shares the same *targetNamespace* Tagged Value as the target package. An *import* element is used where the *targetNamespaces* differ).
Class	A root-level element declaration and *complexType* definition are generated. The element name and type are the same as the Class name. An XSD sequence model group is generated to contain UML attributes generated as elements.
Attribute	An element is declared for each Class attribute. The element name is set to that of the UML attribute name. This is prefixed with the Class name to make the element unique. The *minOccurs* and *maxOccurs* attributes are set to reflect the attribute cardinality. **Note:** If left unspecified, *minOccurs* and *maxOccurs* default to **1**. If the attribute refers to another Class, the element declaration is followed a *complexType* definition, which contains a reference to the appropriate *complexType*.
Association	An element is declared for each association owned by a Class. The element name is set to that of the association role. The *minOccurs* and *maxOccurs* reflect the cardinality of the association. **Note:** If the direction of the association is unspecified, the owner is assumed to be the source.
Generalization (Inheritance)	For single inheritances, an extension element is generated with the base attribute set to the base Classname. The UML attributes of the child Class are then appended to an all model group within the extension element.

UML Construct	Default XSD Production Rules
«enumeration» (stereotype)	A *simpleType* element is declared for the enumeration Class with the name attribute set to the Classname. A restriction element is generated with base set to string. Each of the Class attributes is appended to the restriction element as XSD enumeration elements with value set to the UML attribute name. Any type specification for the UML attributes is ignored by the schema generator.

2.1.2 Generate XSD

The *Generate XML Schema* feature forward engineers a UML Class model to a W3C XML Schema (XSD) file. An XML schema corresponds to a UML package in Enterprise Architect, therefore XML schema generation is a package-level operation. To generate an XML schema from a package, follow the steps below:

1. In the Project Browser, right-click on the package to be converted to XSD. The context menu displays.
2. Select the **Code Engineering | Generate XML Schema** menu option. The Generate XML Schema dialog displays, showing the name of the selected package in the **Source Package** field.


Source Package: ClassLibrary2
Encoding: ISO-8859-1

XSD Style
[✓] Generate global element for all global ComplexTypes

Referenced Package Options
[✓] Generate XSD for Referenced packages
[ ] Prompt when missing Filename

Child Package Options
[✓] Generate XSD for Child packages
[✓ Include all packages] Include <XSDschema> packages

Package	Filename
✓ ClassLibrary2	ClasslibSchema.xsd

3. In the **Encoding** field, set the required XML encoding.
4. In the XSD Style panel, the **Generate global element for all global ComplexTypes** checkbox is selected by default to generate schema in the *Garden of Eden style* 101.
5. In the Referenced Package Options panel, select the:
   - **Generate XSD for Referenced packages** checkbox to generate schema for packages that are referenced by any of the packages selected in the list box
   - **Prompt when missing Filename** checkbox to enable Enterprise Architect to prompt for a filename for a referenced package during schema generation, if the filename is missing.
6. In the Child Package Options panel, select the:
   - **Generate XSD for Child Packages** checkbox to generate schema for child packages of the selected package
   - **Include all packages** radio button to list all child packages under the parent package in the list box
   - **Include <XSDschema> packages** radio button to list only those packages that have the stereotype *«XSDschema»*.

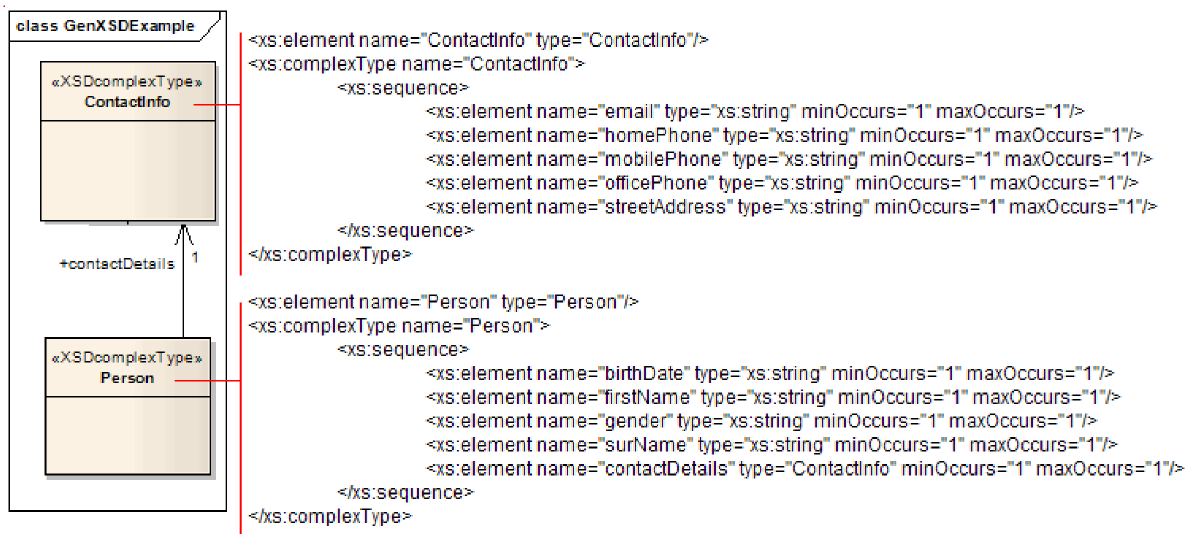
   The list box displays, for each package, the package name and the file path where the schema file is to be generated.
7. If it is necessary to change the file path for a package, double-click on the entry in the list box and, on the Select XML File dialog, type or select the appropriate file path.
8. Ensure that the checkbox is selected for each package required for generation.
9. Click on the **Generate** button to generate the schema for each of the selected packages.
10. The progress of the schema generator is shown in the **Progress** box.
11. When schema generation is complete, click on an entry in the list box and click on the **View Schema** button to review the generated schema.

Tip:

The Generate XML Schema dialog can also be accessed from the active diagram by selecting the **Project | XML Schema | Generate XML Schema** menu option.

2.1.2.1 Generate Global Element

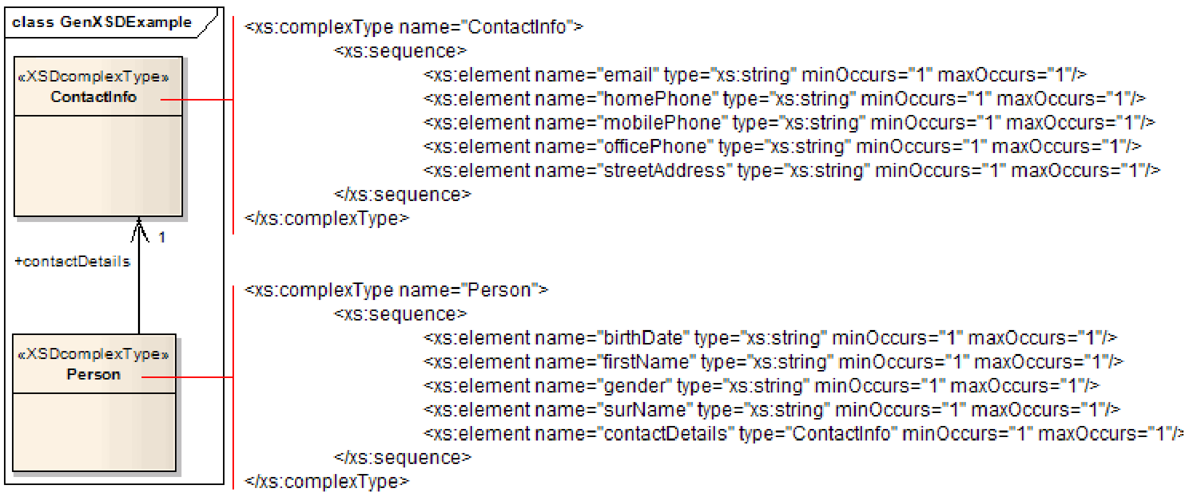
Enterprise Architect, by default, generates XML Schema in the *Garden of Eden* style. For every global *XSDcomplexType* stereotyped Class, Enterprise Architect generates a global element. For example, the following model by default generates the XSD shown:



```
<xs:element name="ContactInfo" type="ContactInfo"/>
<xs:complexType name="ContactInfo">
        <xs:sequence>
                <xs:element name="email" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="homePhone" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="mobilePhone" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="officePhone" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="streetAddress" type="xs:string" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
</xs:complexType>

<xs:element name="Person" type="Person"/>
<xs:complexType name="Person">
        <xs:sequence>
                <xs:element name="birthDate" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="firstName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="gender" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="surName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                <xs:element name="contactDetails" type="ContactInfo" minOccurs="1" maxOccurs="1"/>
        </xs:sequence>
</xs:complexType>
```

You can change this default behaviour by deselecting the **Generate global element for all global ComplexTypes** checkbox on the Generate XML Schema 100 dialog. Then, the generated XSD no longer contains the global element, as shown below:



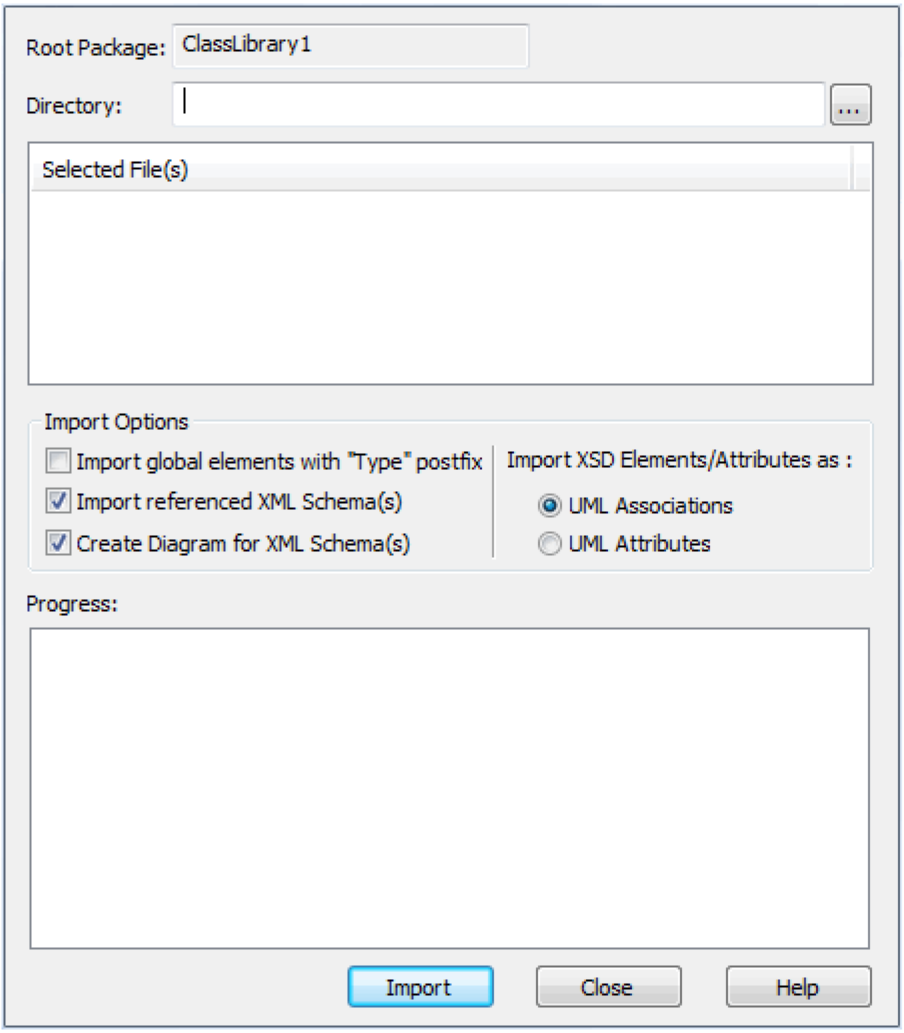
```
<xs:complexType name="ContactInfo">
            <xs:sequence>
                        <xs:element name="email" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="homePhone" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="mobilePhone" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="officePhone" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="streetAddress" type="xs:string" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
</xs:complexType>

<xs:complexType name="Person">
            <xs:sequence>
                        <xs:element name="birthDate" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="firstName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="gender" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="surName" type="xs:string" minOccurs="1" maxOccurs="1"/>
                        <xs:element name="contactDetails" type="ContactInfo" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
</xs:complexType>
```

2.1.3 Import XSD

The **XML Schema Import** facility is used to reverse engineer a W3C XML Schema (XSD) file as a UML Class model. An XSD file is imported into Enterprise Architect as a UML package. To import an XSD file, follow the steps below:

1. In the Project Browser, right-click on the package to contain the imported XSD package. The context menu displays.
2. Select the **Code Engineering | Import XML Schema** menu option. The Import XML Schema dialog displays.



Import Options:
- [ ] Import global elements with "Type" postfix
- [✓] Import referenced XML Schema(s)
- [✓] Create Diagram for XML Schema(s)

Import XSD Elements/Attributes as : [✓ UML Associations] UML Attributes

3. In the **Directory** field, click on the **[...]** (Browse) button. The Select XML Schema(s) dialog displays.
4. Click on the required input file. To select several individual files, press **[Ctrl]** as you click on each one. To select a range of files, press **[Shift]** and click on the first and last file in the range.
5. Click on the **Open** button to return to the Import XML Schema dialog, which now shows the selected files in the **Selected File(s)** field.
6. The **Import global elements with "Type" postfix** 104 checkbox defaults to unselected to import a global element, and the *ComplexType* to which it refers, as a single ComplexType Class.
7. The **Import referenced XML Schema(s)** checkbox defaults to selected, to import any other Schema file referenced by the selected input XML Schema file or files.

   Note:

   If an XML Schema file being imported already exists in the model, Enterprise Architect skips importing the file.

8. The **Create Diagram for XML Schema(s)** checkbox defaults to selected, to display the imported elements on the diagram. If necessary, deselect the checkbox.
9. For the **Import XSD Elements/Attributes as:** field, select the appropriate radio button to import elements and attributes in the XML Schema as:
   - UML Association connectors or
   - UML Class attributes.
10. Click on the **Import** button to import the schema.
11. The progress of the schema import is shown in the **Progress** status bar.

Tip:

The Import XML Schema dialog can also be accessed for the active diagram by selecting the **Project | XML Schema | Import XML Schema** menu option.

Note:

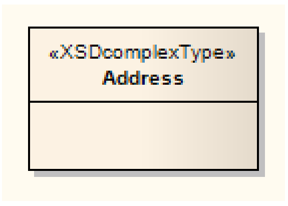
Enterprise Architect uses the *schemaLocation* attribute in the Import and Include elements of an XML Schema to determine the dependencies between the files. Ensure that this attribute is set to a valid file path (and not a URL) for the dependent XML Schema(s) to be imported correctly.

2.1.3.1 Global Element and ComplexType

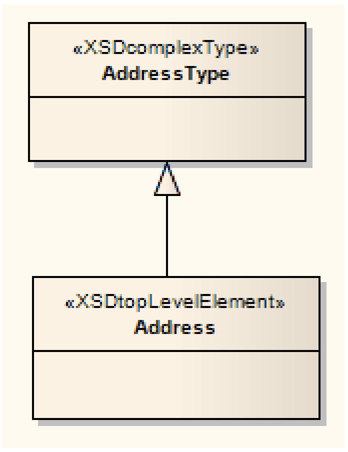
Some XML Schemas have *ComplexType* elements with the same name as the referring global elements, but with the suffix *Type* as shown below:

```
<xs:element name="Address" type="AddressType"/>
<xs:complexType name="AddressType">
        <xs:sequence/>
</xs:complexType>
```

On XSD import, Enterprise Architect treats this global element and its bounding ComplexType as a single entity and creates a single *XSDcomplexType* stereotyped Class with the same name as the global element as shown below:



You can change this default behaviour by selecting the **Import global elements with "Type" postfix** checkbox. When you select this option, Enterprise Architect treats the global element and the ComplexType it is referring to as two separate entities. So, for the above example, Enterprise Architect creates an *XSDtopLevelElement* stereotyped Class for the global element and an *XSDcomplexType* stereotyped Class for the ComplexType, and connects them as follows:



Note:

Enterprise Architect treats the following as two separate entities irrespective of whether the **Import global elements with "Type" postfix** checkbox is selected or unselected:

```
<xs:element name="HomeAddress" type="AddressType"/>
<xs:complexType name="AddressType">
        <xs:sequence/>
</xs:complexType>
```

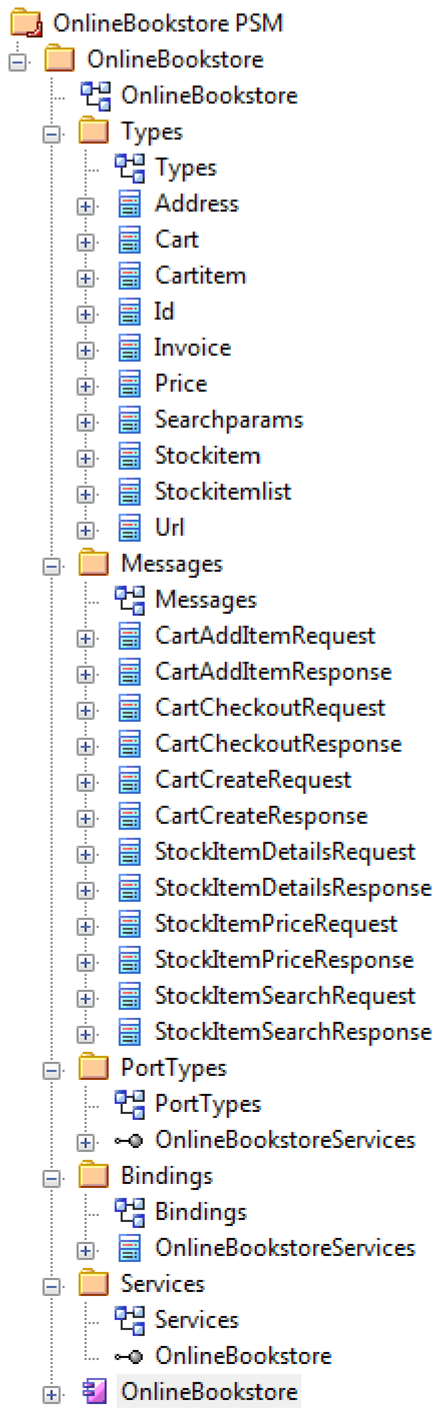
2.2 Web Services (WSDL)

Enterprise Architect supports Forward and Reverse Engineering of the W3C Web Service Definition Language (WSDL). The following topics explain how to use Enterprise Architect to model, generate and import WSDL files:

- Model WSDL 106
- Import WSDL 119
- Generate WSDL 117

2.2.1 Model WSDL

The WSDL pages of the Enterprise Architect UML Toolbox (see *Using Enterprise Architect - UML Modeling Tool*) can be used to conveniently model WSDL documents. WSDL documents are represented as components marked with the stereotype *WSDL*. WSDL documents are contained in a package hierarchy representing the target WSDL namespace and its constituent *XSD Types, Messages, PortTypes, Bindings* and *Services*. The top-level package is stereotyped as a *WSDLnamespace*. The figure below shows a WSDL namespace package structure:



```
OnlineBookstore PSM
  OnlineBookstore
    OnlineBookstore
    Types
      Types
      Address
      Cart
      Cartitem
      Id
      Invoice
      Price
      Searchparams
      Stockitem
      Stockitemlist
      Url
    Messages
      Messages
      CartAddItemRequest
      CartAddItemResponse
      CartCheckoutRequest
      CartCheckoutResponse
      CartCreateRequest
      CartCreateResponse
      StockItemDetailsRequest
      StockItemDetailsResponse
      StockItemPriceRequest
      StockItemPriceResponse
      StockItemSearchRequest
      StockItemSearchResponse
    PortTypes
      PortTypes
      OnlineBookstoreServices
    Bindings
      Bindings
      OnlineBookstoreServices
    Services
      Services
      OnlineBookstore
    OnlineBookstore
```

A *WSDLnamespace* package can contain one or more WSDL components. Each WSDL component can be automatically generated to a WSDL file using Enterprise Architect's built in WSDL generator 117. The following topics describe the various WSDL elements and features supported by Enterprise Architect:

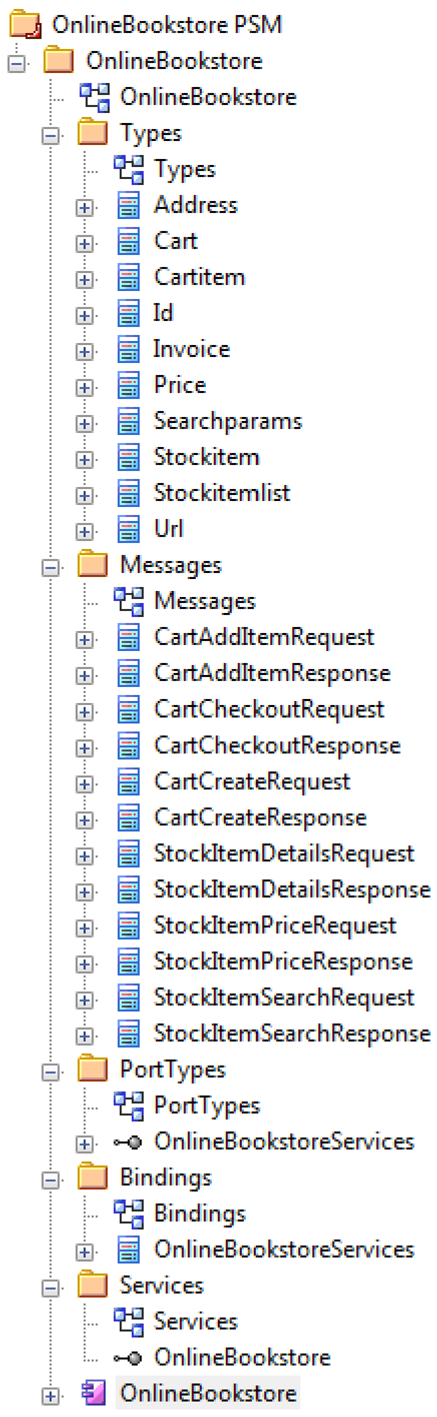
- WSDL Namespace 108
- WSDL Document 110
- WSDL Service 111
- WSDL Port Type 112
- WSDL Message 113
- WSDL Binding 113
- WSDL Port Type Operation 115

- WSDL Message Part 116

2.2.1.1 WSDL Namespace

The WSDL namespace in Enterprise Architect represents the top-level container for the WSDL elements, including WSDL documents. Conceptually it maps to the *targetNamespace* in a WSDL definition element. A given WSDL namespace can reuse its schema Types, Messages, Port Types, Bindings and Service across multiple physical WSDL documents.

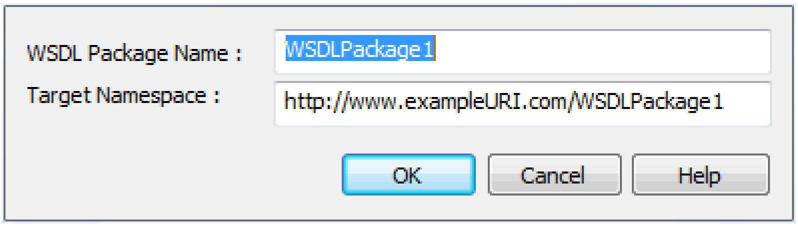
The figure below shows an example WSDL namespace (*OnlineBookstore PSM*, which has a red margin to the bottom right corner), including a single WSDL document:



```
OnlineBookstore PSM
  OnlineBookstore
    OnlineBookstore
    Types
      Types
      Address
      Cart
      Cartitem
      Id
      Invoice
      Price
      Searchparams
      Stockitem
      Stockitemlist
      Url
    Messages
      Messages
      CartAddItemRequest
      CartAddItemResponse
      CartCheckoutRequest
      CartCheckoutResponse
      CartCreateRequest
      CartCreateResponse
      StockItemDetailsRequest
      StockItemDetailsResponse
      StockItemPriceRequest
      StockItemPriceResponse
      StockItemSearchRequest
      StockItemSearchResponse
    PortTypes
      PortTypes
      OnlineBookstoreServices
    Bindings
      Bindings
      OnlineBookstoreServices
    Services
      Services
      OnlineBookstore
    OnlineBookstore
```

To create a new WSDL namespace in your model, follow the steps below.

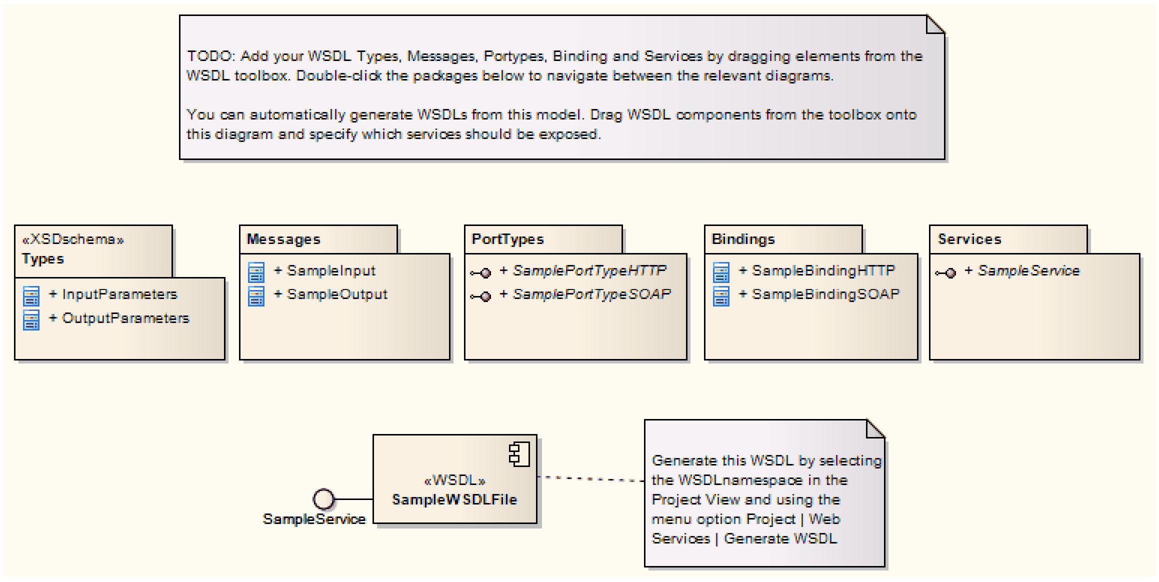
1. Open or create the appropriate diagram.
2. Select the **More Tools | WSDL** menu option from the Enterprise Architect UML Toolbox.
3. Drag the *Namespace* element from the Toolbox onto the diagram. The WSDL Namespace Properties dialog displays:



WSDL Package Name : WSDLPackage1

Target Namespace : http://www.exampleURI.com/WSDLPackage1

4. Type in a **WSDL Package Name** and **Target Namespace** name. You can edit these values later.
5. Click on the **OK** button to create a package stereotyped as *WSDLnamespace*. This contains the following sub-packages and an Overview diagram to navigate between the sub-packages:
   - **Types**: Contains the XSD types used by the WSDL *Message* elements; this package is modeled as an XML Schema 89, and you drag XSDelement 94, XSDsimpleType 93 and XSDcomplexType 93 elements onto the Types diagram from the XML Schema page of the Enterprise Architect UML Toolbox (see *Using Enterprise Architect - UML Modeling Tool*)
   - **Messages**: Contains the WSDL *Messages*, modeled as UML Classes marked with the stereotype *WSDLmessage*
   - **PortTypes**: Contains the WSDL *Port Types*, modeled as UML interfaces marked with the stereotype *WSDLportType*
   - **Bindings**: Contains the WSDL *Bindings*, modeled as UML Classes that realize the *PortTypes*
   - **Services**: Contains the WSDL *Services,* modeled as UML interfaces with associations to each exposed *Binding*.
6. Use the Overview diagram to navigate between the subpackages, by double-clicking the relevant packages. You can edit the sample WSDL elements created in the previous step, or drag new items from the WSDL pages of the Toolbox onto the relevant diagrams.



TODO: Add your WSDL Types, Messages, Portypes, Binding and Services by dragging elements from the WSDL toolbox. Double-click the packages below to navigate between the relevant diagrams.

You can automatically generate WSDLs from this model. Drag WSDL components from the toolbox onto this diagram and specify which services should be exposed.

Generate this WSDL by selecting the WSDLnamespace in the Project View and using the menu option Project | Web Services | Generate WSDL

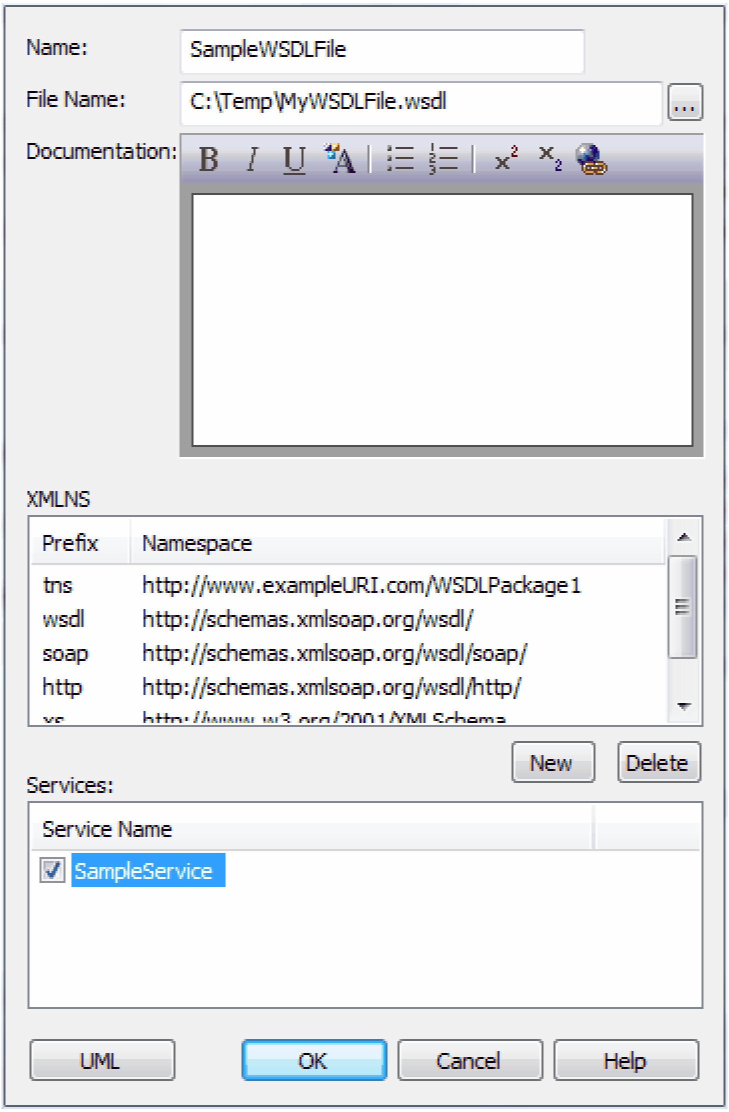
You can edit the WSDL-specific properties of the namespace later by double-clicking the package in the Project Browser. Alternatively, on the WSDL Namespace Properties dialog, click on the **UML** button to invoke the standard Properties dialog for a package. (This button does not display on the initial WSDL Namespace Properties dialog for a new Namespace element.)

2.2.1.2 WSDL Document

WSDL documents are represented in Enterprise Architect by UML components stereotyped as «*WSDL*». These components are modeled as direct child elements of the top-level WSDL namespace package. You can create multiple WSDL documents for a single namespace, thus enabling the services for that namespace to be reused and exposed as required across multiple WSDLs.

To define new WSDL document components for your namespace, follow the steps below:

1. Open the Overview diagram defined for your WSDL namespace package, and drag the *WSDL* element from the Toolbox onto the diagram. The WSDL Document Properties dialog displays.



Name: SampleWSDLFile

File Name: C:\Temp\MyWSDLFile.wsdl

Documentation:

XMLNS

Prefix	Namespace
tns	http://www.exampleURI.com/WSDLPackage1
wsdl	http://schemas.xmlsoap.org/wsdl/
soap	http://schemas.xmlsoap.org/wsdl/soap/
http	http://schemas.xmlsoap.org/wsdl/http/
xs	http://www.w3.org/2001/XMLSchema

Services:

Service Name
☑ SampleService

2. Type in the **Name** and **File Name** for the document.
3. The XMLNS panel lists the default XML namespaces used by the document. If required, click on the **New** button to add further namespaces.

Note:

You can also delete any namespace entries that you add. It is recommended that you do not delete any of the default entries, as it may cause an invalid WSDL document to be generated.

4. Select one or more services that should be exposed by this document. The list of available services is populated from the Services package 111.
5. Click on the **OK** button.

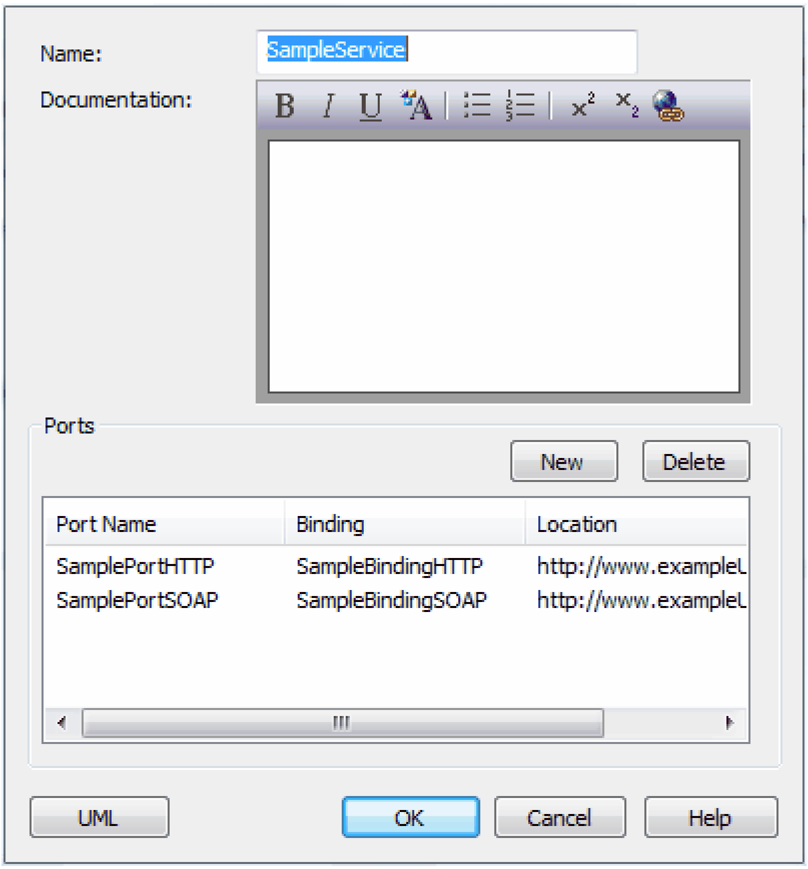
You can edit the WSDL-specific properties of the document later by double-clicking the component in the diagram or the Project Browser. Alternatively, click on the **UML** button in the WSDL Document Properties dialog to invoke the standard Properties dialog for a package. (This button does not display on the initial WSDL Document Properties dialog for a new WSDL element.)

2.2.1.3 WSDL Service

WSDL services are represented in Enterprise Architect by UML interfaces, stereotyped as *WSDLservice*. Services should be defined under the Services packages in the WSDL namespace structure.

To define new *WSDLservice* elements for your namespace, follow the steps below:

1. Open the Overview diagram defined for your WSDL namespace package, and double-click on the *Services* package element to open the Services diagram.
2. Drag the *Service* element from the Toolbox onto the diagram. The WSDL Service dialog displays.



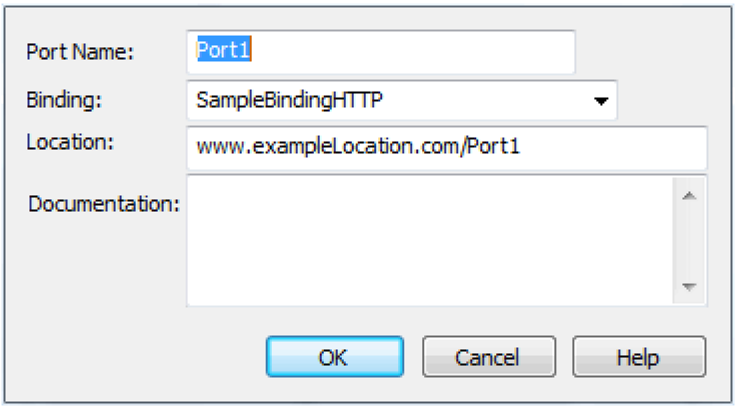
Name: SampleService

Documentation:

Ports

Port Name	Binding	Location
SamplePortHTTP	SampleBindingHTTP	http://www.exampleU
SamplePortSOAP	SampleBindingSOAP	http://www.exampleU

3. In the **Name** field, type the service name.
4. Click on the **New** button to add Service Ports. The WSDL Port dialog displays.


Port Name: Port1
Binding: SampleBindingHTTP
Location: www.exampleLocation.com/Port1
Documentation:

5. Type in the **Port Name** and **Location,** and select a **Binding**. The list of Bindings is taken from those defined in the Bindings package 113.
6. Click on the **OK** button to close the WSDL Port dialog. For each Port defined in this way, Enterprise Architect creates an Association relationship between the Service and corresponding *Binding* element.
7. Click on the **OK** button to close the WSDL Service dialog.

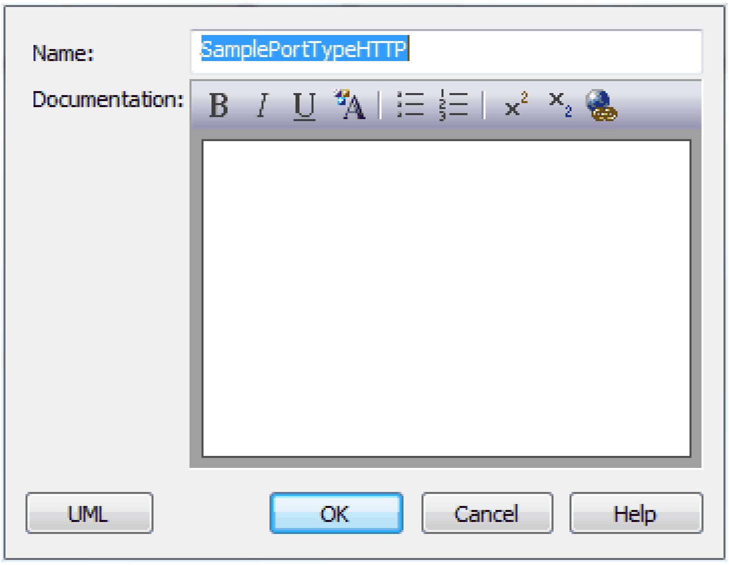
You can edit the WSDL-specific properties of the service later by double-clicking the Service interface in the diagram or Project Browser. Alternatively, click on the **UML** button in the WSDL Service dialog to invoke the standard Properties dialog for an interface. (This button does not display on the initial WSDL Service dialog for a new Service element.)

2.2.1.4 WSDL Port Type

WSDL *Port Types* are represented in Enterprise Architect by UML interfaces stereotyped as *WSDLportType*. PortTypes should be defined under the *PortTypes* packages in the WSDL namespace structure.

To define new WSDLportType elements for your namespace, follow the steps below:

1. Open the Overview diagram defined for your WSDL namespace package, and double-click on the PortTypes package to open the PortTypes diagram.
2. Drag the *Port Type* element from the Toolbox onto the diagram. The WSDL PortType dialog displays.


Name: SamplePortTypeHTTP
Documentation:

3. Type in the name for the portType.
4. Click on the **OK** button to close the WSDL PortType dialog.
5. Define operations for the portType by dragging the Port Type Operation 115 item from the WSDL page of the Enterprise Architect UML Toolbox onto the portType interface.

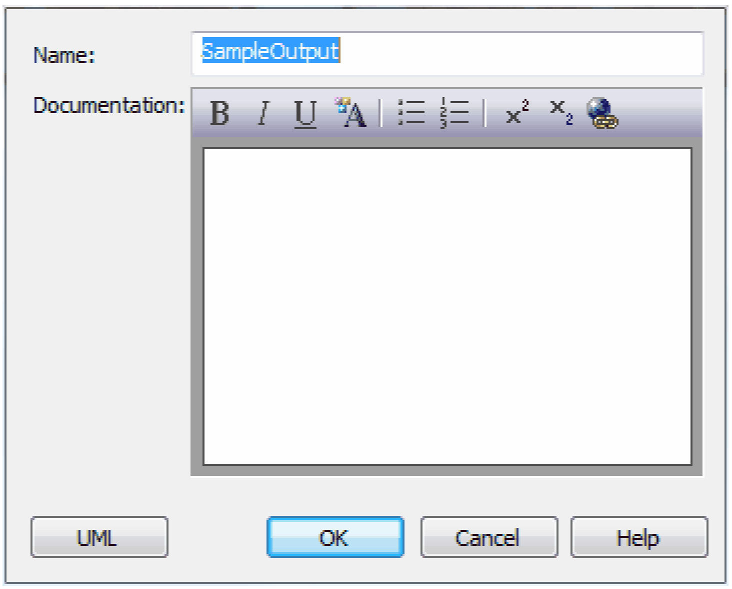
You can edit the WSDL-specific properties of the portType later by double-clicking the interface in the diagram or Project Browser. Alternatively, in the WSDL PortType dialog, click on the **UML** button to invoke the standard Properties dialog for an interface. (This button does not display on the initial WSDL PortType dialog for a new PortType element.)

2.2.1.5 WSDL Message

WSDL messages are represented in Enterprise Architect by UML Classes stereotyped as *WSDLmessage*. Messages should be defined under the *Messages* package in the WSDL namespace structure.

To define new WSDLmessage elements for your namespace, follow the steps below:

1. Open the Overview diagram defined for your WSDL namespace package, and double-click on the Messages package to open the Messages diagram.
2. Drag the *Message* element from the Toolbox onto the diagram. The WSDL Message dialog displays.



3. Type in the **Name** for the message.
4. Click on the **OK** button to close the WSDL Message dialog.
5. You can define parts for the message by dragging the *Message Part* 116 element from the WSDL Elements page of the Enterprise Architect UML Toolbox onto the Message element.

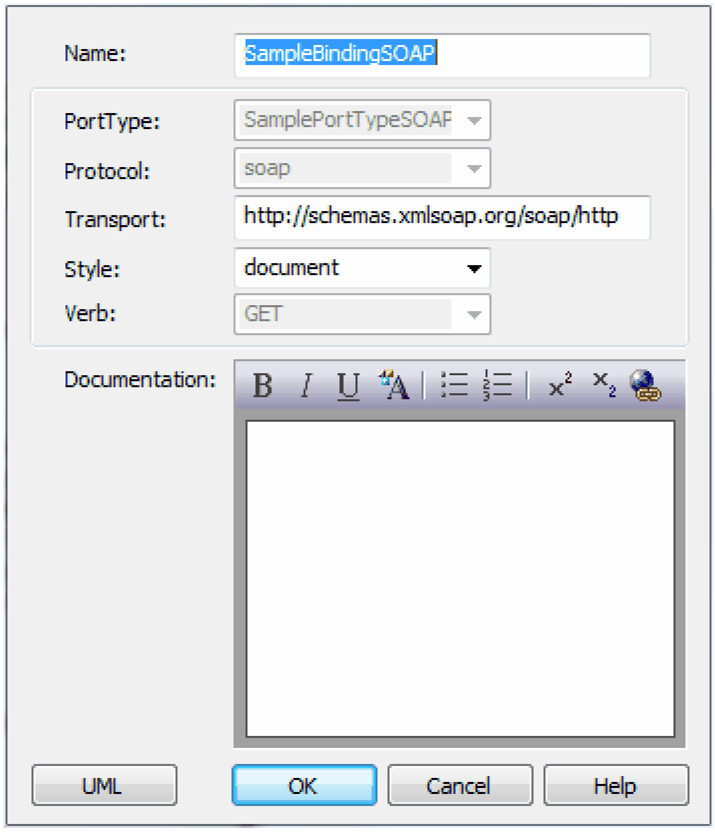
You can edit the WSDL-specific properties of the message later by double-clicking the Message element in the diagram or Project Browser. Alternatively, on the WSDL Message dialog, click on the **UML** button to invoke the standard Properties dialog for a Class. (This button does not display on the initial WSDL Message dialog for a new Message element.)

2.2.1.6 WSDL Binding

WSDL bindings are represented in Enterprise Architect by UML Classes stereotyped as *WSDLbinding*. Bindings should be defined under the Bindings package in the WSDL namespace structure. Each *WSDLbinding* Class implements the operations specified by a particular *WSDLportType* interface. Therefore, WSDLportTypes should be defined before 112 creating *WSDLbindings*.

To define new *WSDLbinding* elements for your namespace, follow the steps below:

1. Open the Overview diagram defined for your WSDL namespace package, and double-click on the Bindings package to open the Bindings diagram.
2. Drag the *Binding* element from the Toolbox onto the diagram. The WSDL Binding dialog displays.


Name: SampleBindingSOAP

PortType: SamplePortTypeSOAP

Protocol: soap

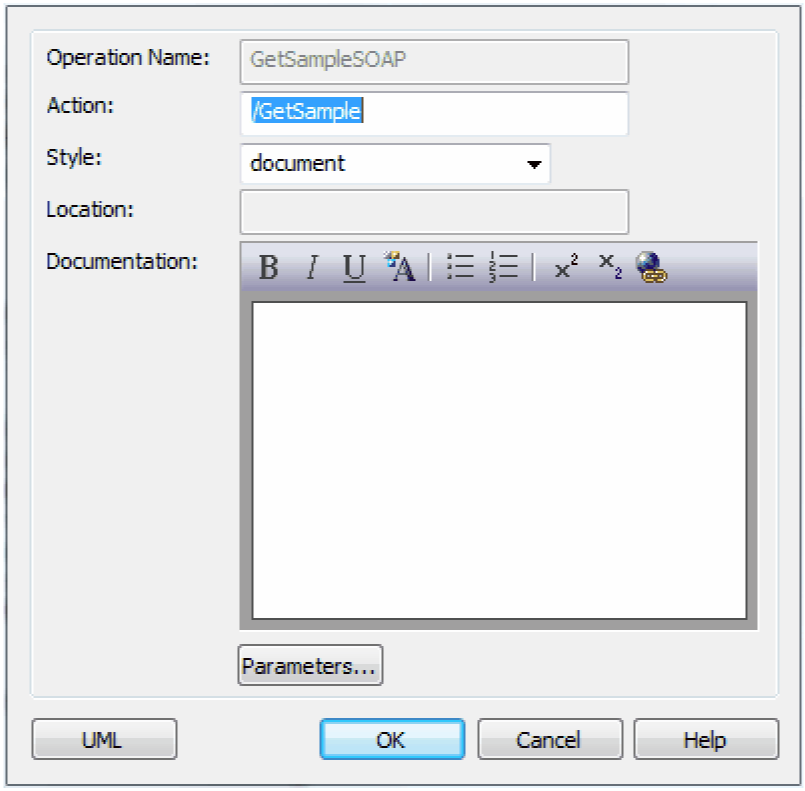
Transport: http://schemas.xmlsoap.org/soap/http

Style: document

Verb: GET

Documentation:

3. Type in a **Name** for the Binding.
4. Select the **PortType** for the Binding; the drop-down list of PortTypes is taken from those defined in the PortTypes package.
5. Select the **Protocol** for the Binding, either **http** or **soap**.
6. For SOAP Bindings, enter the **Transport** URL and select the **Style**. For http Bindings, select the **Verb**.
7. Click on the **OK** button to close the WSDL Binding dialog and create the binding. A *realization* connector is created between the binding and the corresponding *Port Type* interface.
8. To specify the Binding operations, select and double-click on an Operation in the Binding element. The WSDL Binding Operation Details dialog displays.


Operation Name: GetSampleSOAP
Action: /GetSample
Style: document
Location:
Documentation:

9. Type in or select the Binding Operation details.
10. Click on the **Parameters** button. The WSDL Binding Operation Parameters dialog displays. For each input, output and fault, click on the **Details** button and enter the details.
11. Click on the **OK** button on each of the WSDL Binding Parameter Details, WSDL Binding Operation Parameters and WSDL Binding Operation Details dialogs to close them.

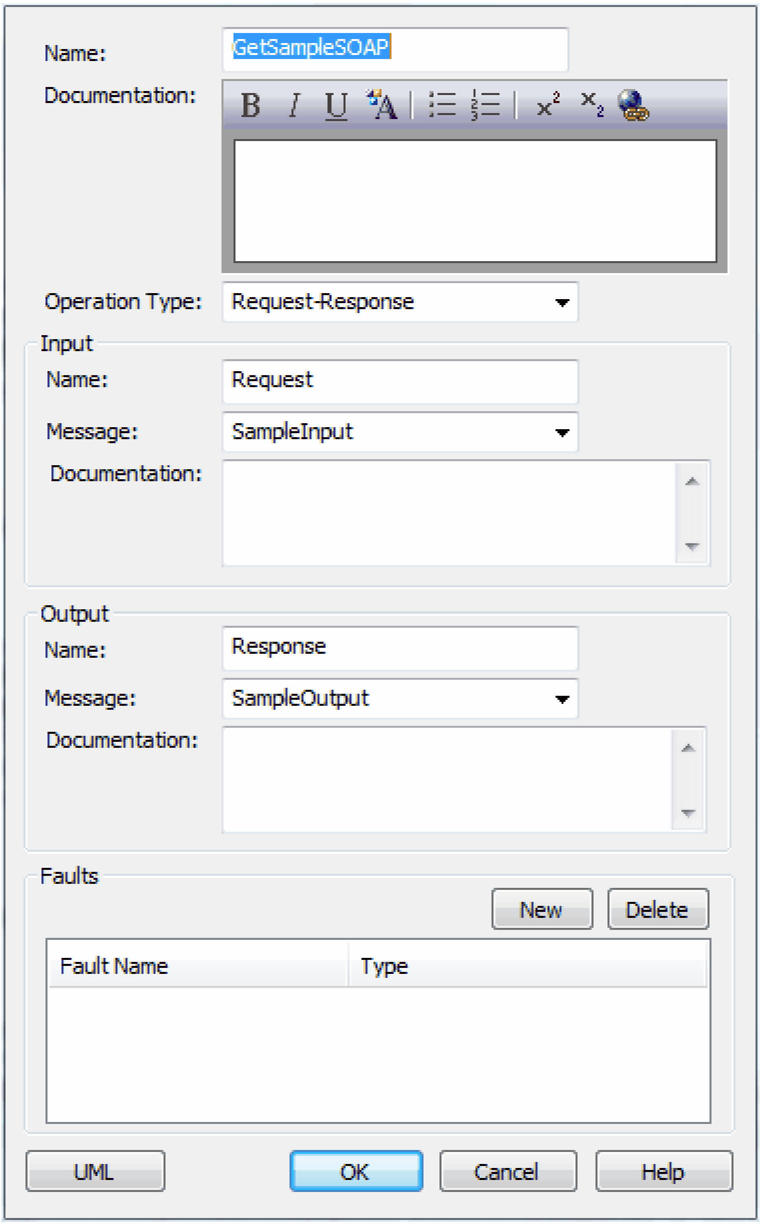
You can edit the WSDL-specific properties of the binding later by double-clicking the binding Class in the diagram or Project Browser. Alternatively, on the WSDL Binding dialog, click on the **UML**button to invoke the standard Properties dialog for a Class. (This button does not display on the initial WSDL Binding dialog for a new Binding element.)

2.2.1.7 WSDL Port Type Operation

WSDL portType operations are represented in Enterprise Architect by operations defined as part of a WSDLportType interface (see the *WSDL Port Type* 112 topic).

To add portType operations to your WSDLportType interfaces, follow the steps below.

1. Open the Overview diagram defined for your WSDL namespace package, and double-click on the PortTypes package to open the PortTypes diagram.
2. Drag the *PortType Operation* item onto a WSDLPortType stereotyped interface. The WSDL PortType Operation dialog displays.


Name: GetSampleSOAP

Documentation:

Operation Type: Request-Response

Input

Name: Request

Message: SampleInput

Documentation:

Output

Name: Response

Message: SampleOutput

Documentation:

Faults

Fault Name	Type

3. Type in the **Name** for the operation.
4. Select the **Operation Type**.
5. Type in or select the **Input**, **Output** and **Fault** details for the operation. The **Message** drop-down list is taken from the WSDLmessage elements defined under the Messages package.
6. Click on the **OK** button to close the WSDL PortType Operation dialog and create the operation.

You can edit the WSDL-specific properties of the portType operation later by double-clicking the operation in the diagram or Project Browser. Alternatively, on the WSDL PortType Operation dialog, click on the **UML** button to invoke the standard Properties dialog for an operation. (This button does not display on the initial WSDL PortType Operation dialog for a new PortType Operation.)

2.2.1.8 WSDL Message Part

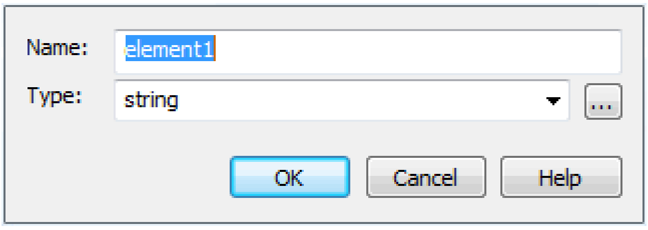
WSDL message parts are represented in Enterprise Architect by UML attributes defined as part of a WSDLmessage Class (see the *WSDL Message* 113 topic).

To add message parts to your WSDLmessage Classes, follow the steps below:

1. Open the Overview diagram defined for your WSDL namespace package, and double-click on the

Messages package to open the Messages diagram.

2. Drag the *Message Part* element onto a WSDLmessage stereotyped Class. The WSDL Message Part dialog displays.



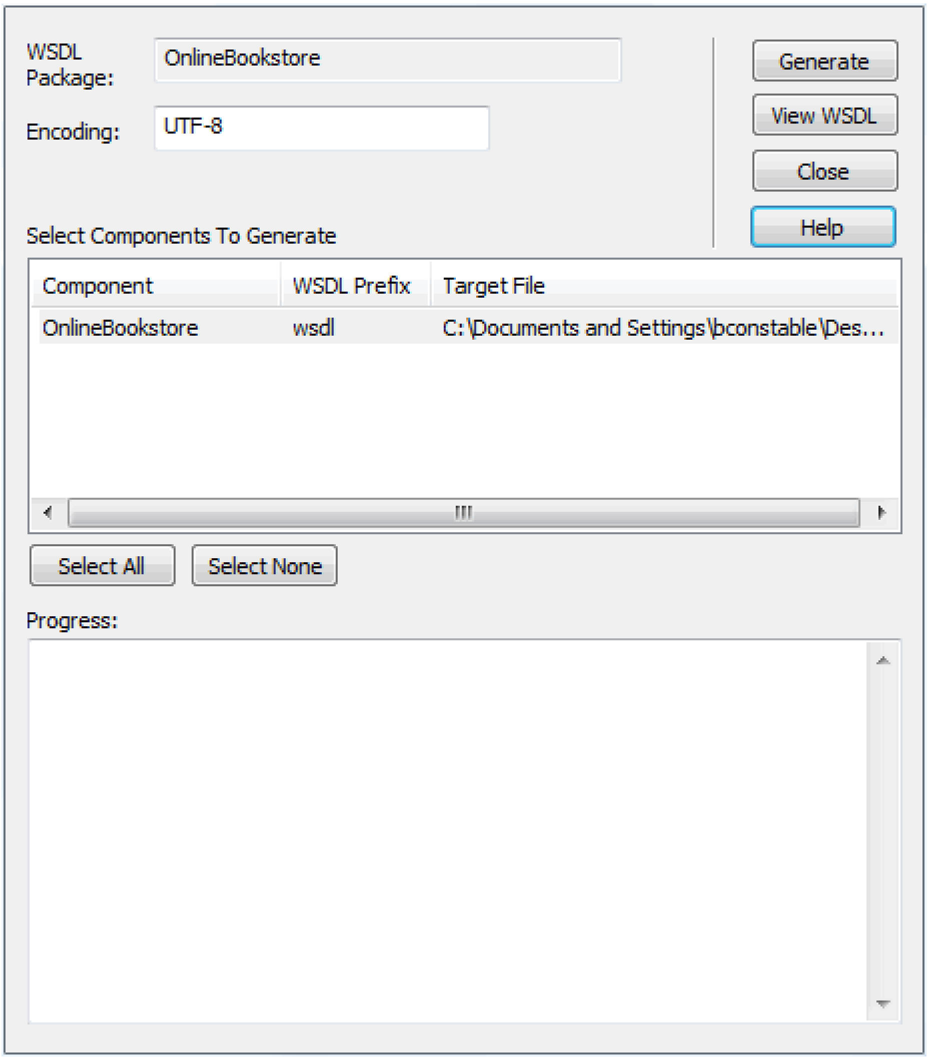
3. Type in a **Name** and **Type** for the message part. The type should be selected from the drop-down list of primitive XSD types or from the types defined under the Types package.
4. Click on the **OK** button.

You can edit the WSDL-specific properties of the message part later by double-clicking the attribute in the diagram or Project Browser. Alternatively, on the WSDL Message Part dialog, click on the **UML** button to invoke the standard Properties dialog for an attribute. (This button does not display on the initial WSDL Message Part dialog for a new message part attribute.)

2.2.2 Generate WSDL

The *Generate WSDL* feature forward engineers a UML model to a Web Service Definition Language (WSDL) file. The Generate WSDL feature acts on a package stereotyped with *WSDLnamespace*. It is used to generate any or all of the WSDL stereotyped components owned by the target *WSDLnamespace* structure. To generate one or more WSDL files from a WSDLnamespace, follow the steps below:

1. In the Project Browser, right-click on the target *WSDLnamespace* package to display the context menu.
2. Select the **Code Engineering | Generate WSDL** menu option. The Generate WSDL dialog displays.



WSDL Package: OnlineBookstore

Encoding: UTF-8

Select Components To Generate

Component	WSDL Prefix	Target File
OnlineBookstore	wsdl	C:\Documents and Settings\bconstable\Des...

Progress:

3. For each WSDL component, set the required output file using the Target File column.
4. Using the **Encoding** field, set the required XML encoding.
5. Click on the **Generate** button to generate the WSDL files.
6. The progress of the WSDL generator is shown in the **Progress** edit box.

Tip:

The Generate WSDL dialog can also be accessed from the active diagram by selecting the **Project | Generate WSDL** menu option.

2.2.3 Import WSDL

The *WSDL Import* facility is used to reverse engineer WSDL files into UML Class models.

Note:

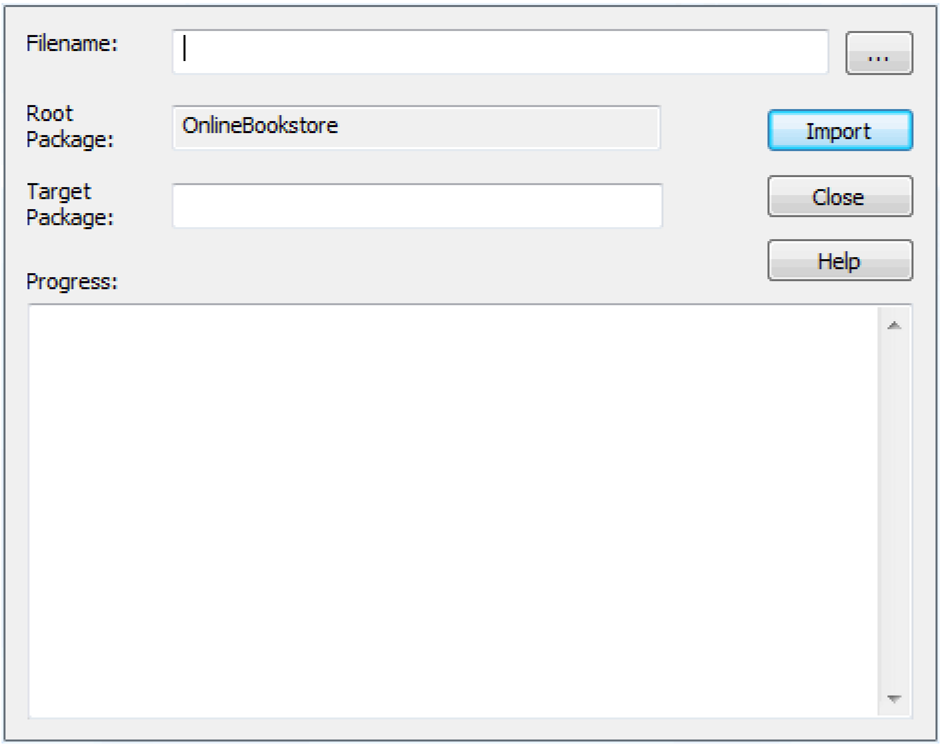
Enterprise Architect cannot import a WSDL file that references WSDL constructs existing outside the target file. For example, Enterprise Architect can import a WSDL as shown in http://www.w3.org/TR/wsdl.html#_example1 but not a file as shown in http://www.w3.org/TR/wsdl.html#_style. Attempting to import the second WSDL file would result in the following error message:

Cannot Import Split Files.

To avoid this limitation, combine the split WSDL files into a single file and then import it into Enterprise Architect.

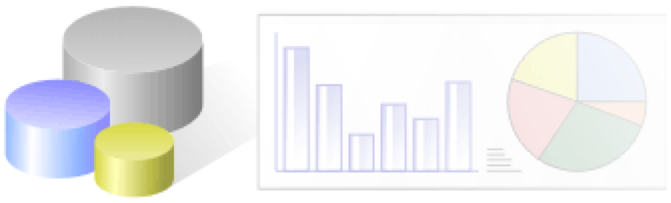
To import a WSDL file, follow the steps below:

1. In the Project Browser, right-click on the package to contain the imported WSDL package. The context menu displays.
2. Select the **Code Engineering | Import WSDL** menu option. The Import WSDL dialog displays.



3. In the **Filename** field, select the input file.
4. The **Target Package** field is automatically set to the name of the selected input file. If required, change this name.
5. Click on the **Import** button to import the schema.
6. The progress of the WSDL import is shown in the Progress status bar.

3 Data Modeling



You perform database modeling and database design in Enterprise Architect using the *UML Data Modeling Profile*. This profile provides easy-to-use and easy-to-understand extensions to the UML standard, mapping the database concepts of tables and relationships onto the UML concepts of Classes and associations. These extensions also enable you to model database keys, triggers, constraints, RI and other relational database features.

Note:

The UML Data Modeling Profile is not currently a ratified standard; however it has wide industry support and is a useful method for bridging the gap between the UML and conventional relational database modeling.

Typical data modeling tasks you might perform are listed at the end of this topic.

Tables and Columns

The basic modeling *structure* of a relational database is the *table*, which represents a set of records, or rows, with the same structure. The basic organizational *element* of a relational database is the *column*. Every individual item of data entered into a relational database is represented by a value in a column of a row in a table.

The UML Data Modeling Profile represents:

- Tables as stereotyped *Classes*; that is, Class elements with a *stereotype* of **table**
- Columns as stereotyped *attributes*; that is, attributes with a *stereotype* of **column**.

Enterprise Architect can generate simple DDL scripts to create the tables in your model. You can also perform Model Driven Architecture (MDA) Transformations to DDL - Enterprise Architect provides a template specifically for DDL transformations (see the *MDA Transformations User Guide*).

To help you map Class attributes to Table fields, you can create connectors between specific attributes in the Class element and the column attributes in the Table element. See the *Connect to Element Feature* topic (see the *Work With Connectors* section of *UML Modeling With Enterprise Architect - UML Modeling Tool*).

Database Keys

Two types of key are used to access tables: *Primary Keys* and *Foreign Keys*. A Primary Key uniquely identifies a record in a table, while a Foreign Key accesses data in some other related table via its Primary Key.

A Primary Key consists of one or more columns; a simple Primary Key (single column) is defined as the attribute of a stereotyped operation. A complex Primary Key (several columns) is defined as the stereotyped operation itself.

A Foreign Key is a collection of columns (attributes) that together have some operational meaning (they enforce a relationship to a Primary Key in another table). Foreign keys are represented in Enterprise Architect as operations with the stereotype **FK**; the operation parameters become the columns involved in the key.

Supported Databases

Enterprise Architect supports import of database schema from these databases:

- DB2
- Firebird/InterBase
- Informix
- Ingres
- MS Access 97, 2000, 2003

- Access 2007
- MS SQL Server 2000, 2005, 2008
- MySQL
- Oracle 9i, 10g and 11g
- PostgreSQL
- Sybase Adaptive Server Anywhere (Sybase ASA)
- Sybase Adaptive Server Enterprise (Sybase ASE).

Notes:

- You can download SQL Server 2005 data types and SQL Server 2008 data types from the Resources page of the Sparx Systems web site.
- Firebird 1.5 database tables can be modeled and generated as InterBase tables. Firebird tables can be imported but are treated as InterBase tables.

Typical Tasks

Typical tasks you can perform when modeling or designing databases include:

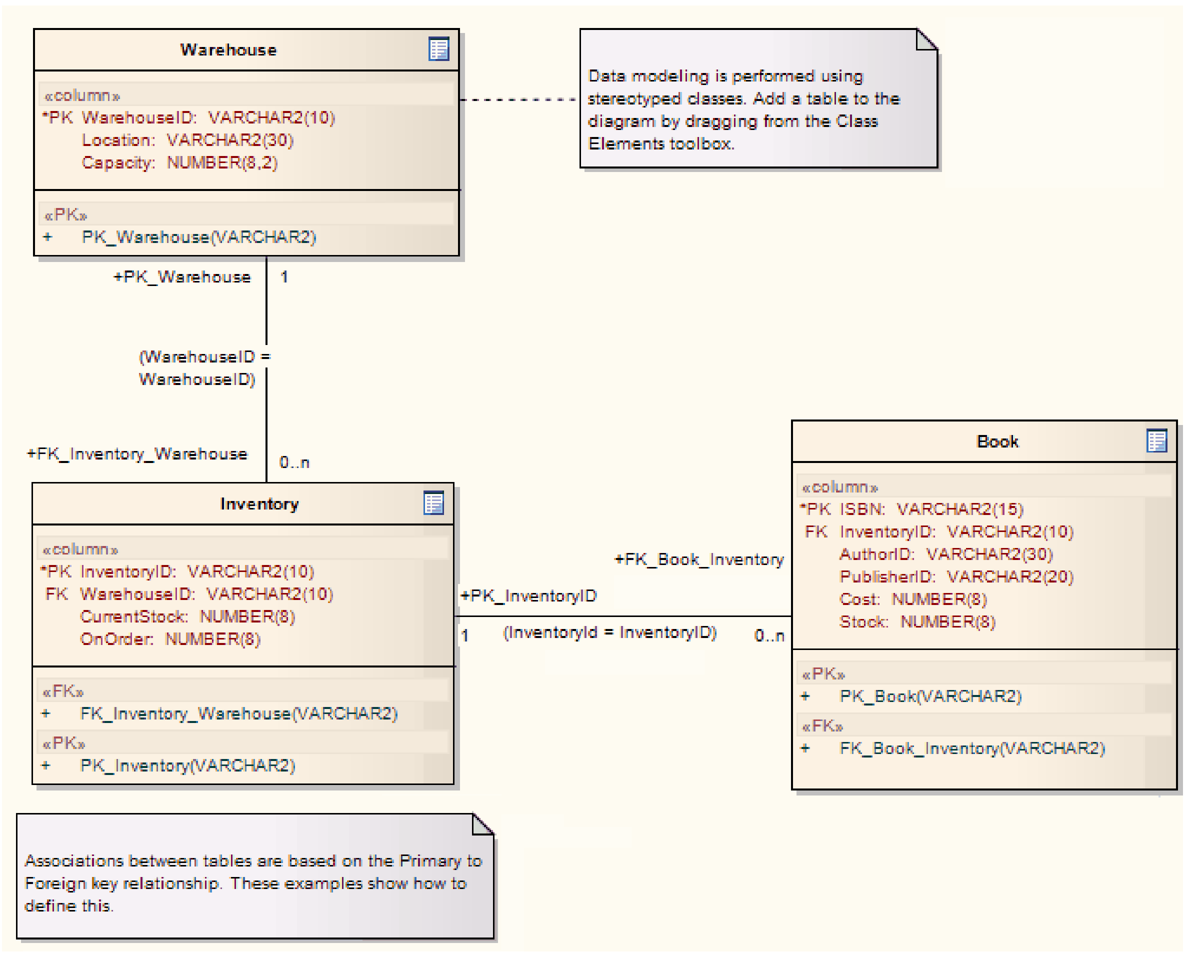
- Create a Data Model Diagram 122
- Create a Table 123
- Set Properties of a Table 124
- Create Columns 130
- Create Oracle Packages 133
- Create Primary Keys 134
- Create Foreign Keys 137
- Create Stored Procedures 143
- Create Views 145
- Create Indexes, Check Constraints and Triggers 147
- Generate DDL for a Table 149
- Generate DDL for a Package 151, and compare with the database
- Convert Datatypes for a Table 155
- Convert Datatypes for a Package 156
- Customize Datatypes for a DBMS 158
- Import a Database Schema from an ODBC Data Source 160

3.1 A Data Model Diagram

An example of a *Data Model* diagram is provided below, showing three tables that are linked on *primary to foreign* key pairs with associated multiplicity.

Note the use of stereotyped operations for Primary (PK) and Foreign (FK) keys. Operations could also be added for:

- Triggers 147
- Constraints 147 (**check**, **unique**)
- Indexes 147



Warehouse
«column»
*PK WarehouseID: VARCHAR2(10)
Location: VARCHAR2(30)
Capacity: NUMBER(8,2)
«PK»
+ PK_Warehouse(VARCHAR2)
Data modeling is performed using stereotyped classes. Add a table to the diagram by dragging from the Class Elements toolbox.
+PK_Warehouse
1
(WarehouseID = WarehouseID)
+FK_Inventory_Warehouse
0..n
Inventory
«column»
*PK InventoryID: VARCHAR2(10)
FK WarehouseID: VARCHAR2(10)
CurrentStock: NUMBER(8)
OnOrder: NUMBER(8)
«FK»
+ FK_Inventory_Warehouse(VARCHAR2)
«PK»
+ PK_Inventory(VARCHAR2)
+FK_Book_Inventory
+PK_InventoryID
1
(InventoryId = InventoryID)
0..n
Book
«column»
*PK ISBN: VARCHAR2(15)
FK InventoryID: VARCHAR2(10)
AuthorID: VARCHAR2(30)
PublisherID: VARCHAR2(20)
Cost: NUMBER(8)
Stock: NUMBER(8)
«PK»
+ PK_Book(VARCHAR2)
«FK»
+ FK_Book_Inventory(VARCHAR2)
Associations between tables are based on the Primary to Foreign key relationship. These examples show how to define this.

A Data Model diagram is represented in Enterprise Architect as a Class diagram, and is created in exactly the same way as other diagrams (see the *Working With Diagrams* topic in *UML Modeling With Enterprise Architect - UML Modeling Tool*).

3.2 Create a Table

What is a Table?

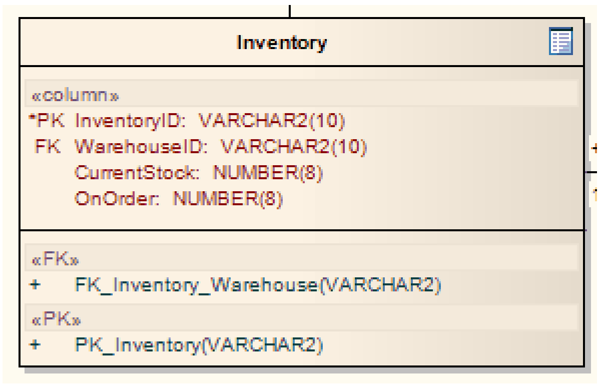
The basic modeling structure of a relational database is the *Table*. A Table represents a set of records, or rows, with the same structure.

The *UML Data Modeling Profile* represents a Table as a stereotyped Class; that is, a Class element with a stereotype of **table** applied to it. A table icon is shown in the upper right corner of the image when it is shown on a Data Model diagram.

Create a Table

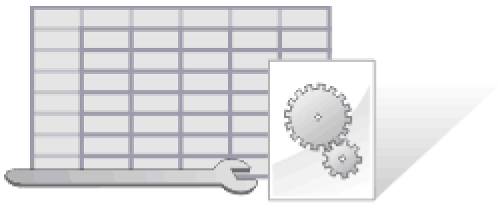
To create a Table, follow the steps below:

1. Select a diagram.
2. Select the **More Tools | Data Modeling** menu option on the Enterprise Architect UML Toolbox.
3. Click on the *Table* element in the list of elements, then click on the diagram. The Table element is displayed on the diagram.


Inventory
«column»
*PK InventoryID: VARCHAR2(10)
FK WarehouseID: VARCHAR2(10)
CurrentStock: NUMBER(8)
OnOrder: NUMBER(8)
«FK»
+ FK_Inventory_Warehouse(VARCHAR2)
«PK»
+ PK_Inventory(VARCHAR2)

4. If the Class: Table n Properties dialog does not display, double-click on the Table to display it.
5. In the **Name** field, type a name for the Table and set any other properties 124 as required.
6. Click on the **OK** button.

3.3 Set Table Properties



Once you have created your table, you can set its properties. Most table properties can be set from the Properties dialog, as described below. However, some properties must be entered as Tagged Values as described for setting the value of the Table Owner 126 and, for MySQL 126 and Oracle 127 databases, setting the table options.

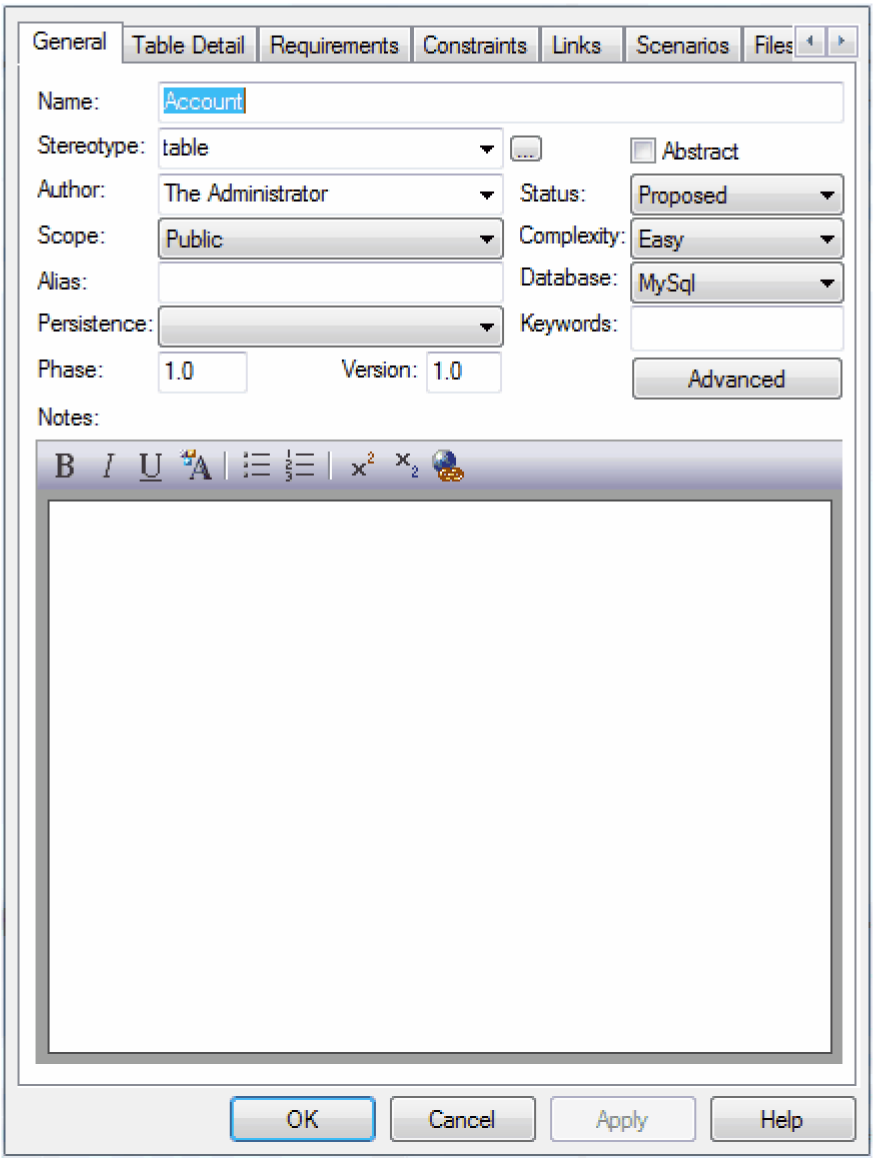
Set the Database Type

The most important property to set for a table (after its name) is the *database type*. This defines the list of datatypes that are available for defining columns, and also declares which dialect of DDL is generated. Enterprise Architect supports the following databases:

- DB2
- Informix
- Ingres
- InterBase
- MS Access 97, 2000, 2003
- Access 2007
- MySQL
- Oracle 9i, 10g and 11g
- PostgreSQL
- SQL Server 2000, 2005 and 2008
- SQLServer7
- Sybase Adaptive Server Anywhere (Sybase ASA)
- Sybase Adaptive Server Enterprise (Sybase ASE).

To set the database type, follow the steps below:

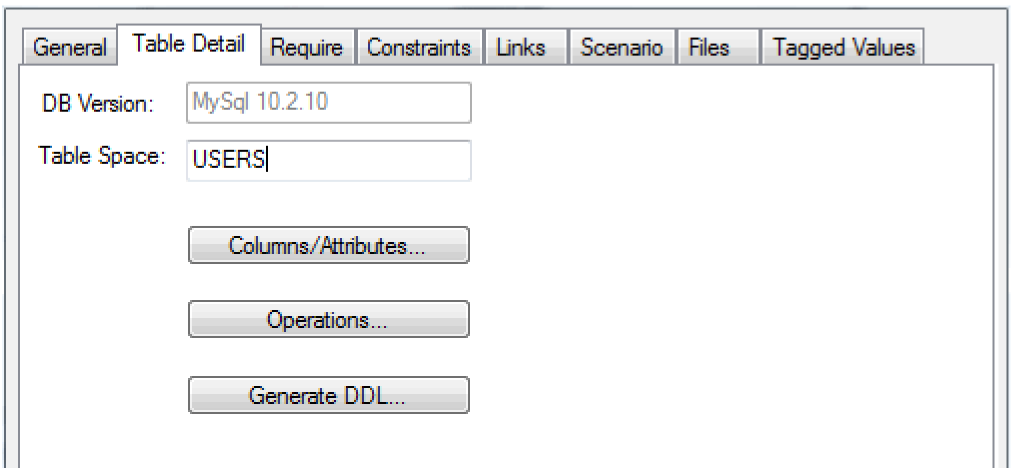
1. Double-click on the table element in a diagram to open the Properties dialog.
2. Select the General tab.



Name: Account
Stereotype: table
Abstract
Author: The Administrator
Status: Proposed
Scope: Public
Complexity: Easy
Alias:
Database: MySql
Persistence:
Keywords:
Phase: 1.0
Version: 1.0
Notes:

3. In the **Database** field, click on the drop-down arrow and select the database type.
4. Click on the **Apply** button to save changes.

By clicking on the Table Detail tab on this dialog, you can access the Columns dialog 130 or Operations dialog 147, or you can Generate DDL 149 for this table.



DB Version: MySql 10.2.10

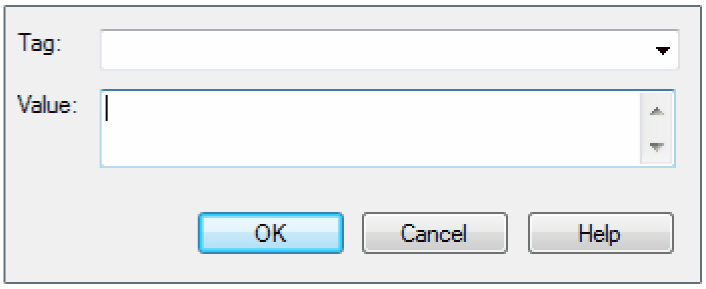
Table Space: USERS

3.3.1 Set Table Owner

To define the owner of a table, follow the steps below:

1. Select the Tagged Values tab of the table Properties dialog. The Tagged Values tab shows the tags for the table.

2. Click on the **New Tag** button. The Tagged Value dialog displays.



3. In the **Tag** field, type the tag name **Owner**. In the **Value** field, type a value for the *Owner* tag.

Note:

For a PostgreSQL database, to define the owner name:

- In the **Tag** field, type the tag name **OWNER TO**
- In the **Value** field, type **Owner_Name**.

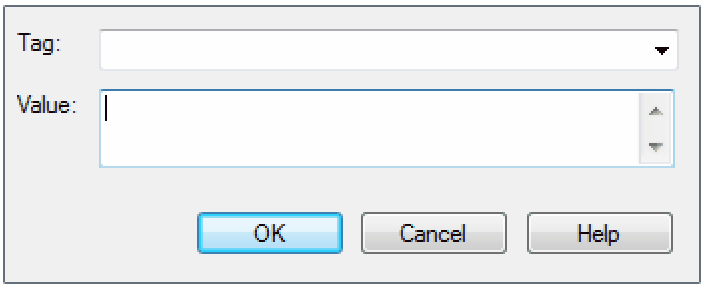
4. Click on the **OK** button to confirm the operation. Generated DDL includes the table owner in the SQL script.

3.3.2 Set MySQL Options

In MySQL, to make use of foreign keys you must declare the table type as *InnoDB*. To do this, follow the steps below:

1. Select the Tagged Values tab of the table Properties dialog. The Tagged Values tab shows the tags for the table.

2. Click on the **New Tag** button. The Tagged Value dialog displays.



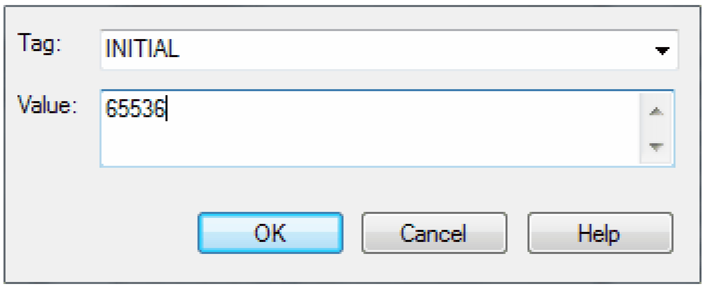
3. In the **Tag** field, type the tag name **Type**. In the **Value** field, type **InnoDB** as the value for the *Type* tag.
4. Click on the **OK** button to confirm the operation. Generated DDL includes the table type in the SQL script.
5. To allow for later versions of MySQL, additional table options that can be added in the same manner include:

Tag	Value (Example)
ENGINE	InnoDB
CHARACTER SET	latin1
CHARSET	latin1
COLLATE	latin1_german2_ci

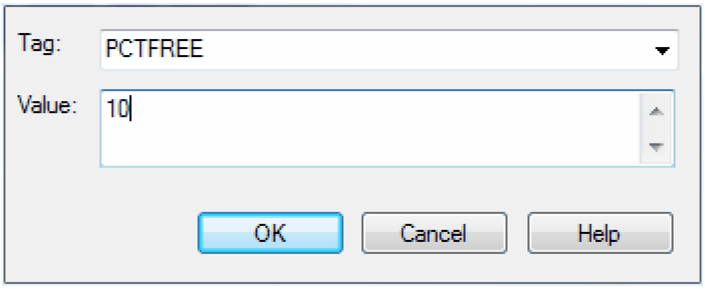
3.3.3 Set Oracle Table Properties

For Oracle, you can set table properties using the table's Tagged Values. Follow the steps below:

1. Select the Tagged Values tab of the table Properties dialog. The Tagged Values tab shows the tags for the table.
2. Click on the **New Tag** button . The Tagged Value dialog displays.
3. Define the table properties as shown in the examples below:



Tag: INITIAL
Value: 65536



Tag: PCTFREE
Value: 10

4. Click on the **OK** button to save the Tagged Value.

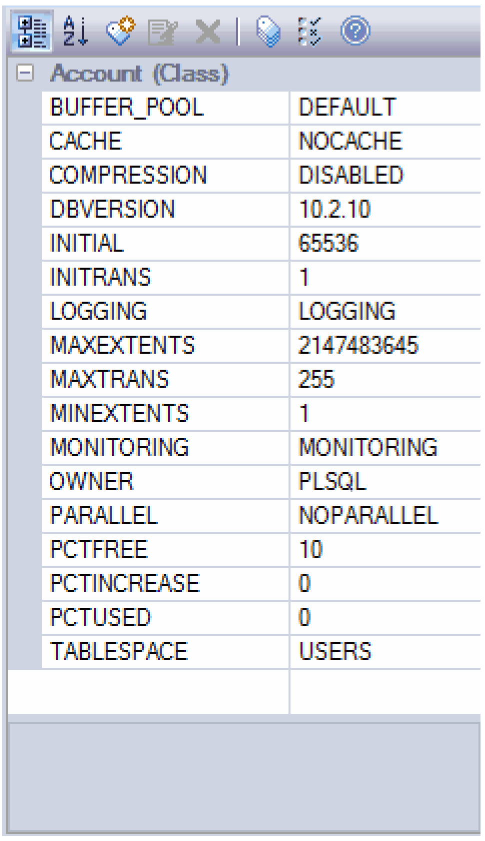
All available properties for an Oracle table are listed below.

Note:

The same properties can be added to indexes and constraints 147. Highlight the index or constraint and add the properties as Tagged Values.

Property/Tag	Value
BUFFER_POOL	DEFAULT
CACHE	NOCACHE
DBVERSION	9.0.111
FREELISTS	1
GRANT OWNER1	SELECT
GRANT OWNER2	DELETE, INSERT, SELECT, UPDATE
INITIAL	65536
INITRANS	1
LOGGING	LOGGING
MAXEXTENTS	2147483645
MAXTRANS	255
MINEXTENTS	1
MONITORING	MONITORING
OWNER	OWNER1
PARALLEL	NOPARALLEL
PCTFREE	10
PCTINCREASE	0
PCTUSED	0
SYNONYMS	PUBLIC:TABLE_PUB;OWNER2:TABLE_OWNER2
TABLESPACE	MY_TABLESPACE
TEMPORARY	YES

The properties defined for a given table are listed on the Tagged Values tab, as illustrated by the following typical Tagged Value list:


Account (Class)

BUFFER_POOL	DEFAULT
CACHE	NOCACHE
COMPRESSION	DISABLED
DBVERSION	10.2.10
INITIAL	65536
INITRANS	1
LOGGING	LOGGING
MAXEXTENTS	2147483645
MAXTRANS	255
MINEXTENTS	1
MONITORING	MONITORING
OWNER	PLSQL
PARALLEL	NOPARALLEL
PCTFREE	10
PCTINCREASE	0
PCTUSED	0
TABLESPACE	USERS

3.4 Create Columns

What is a Column?

The basic organizational element of a relational database is the *column*. Every individual item of data entered into a relational database is represented as a value in a column of a row in a table. Columns are represented in the UML Data Modeling Profile as a stereotyped attribute; that is, an attribute with the *Column* stereotype.

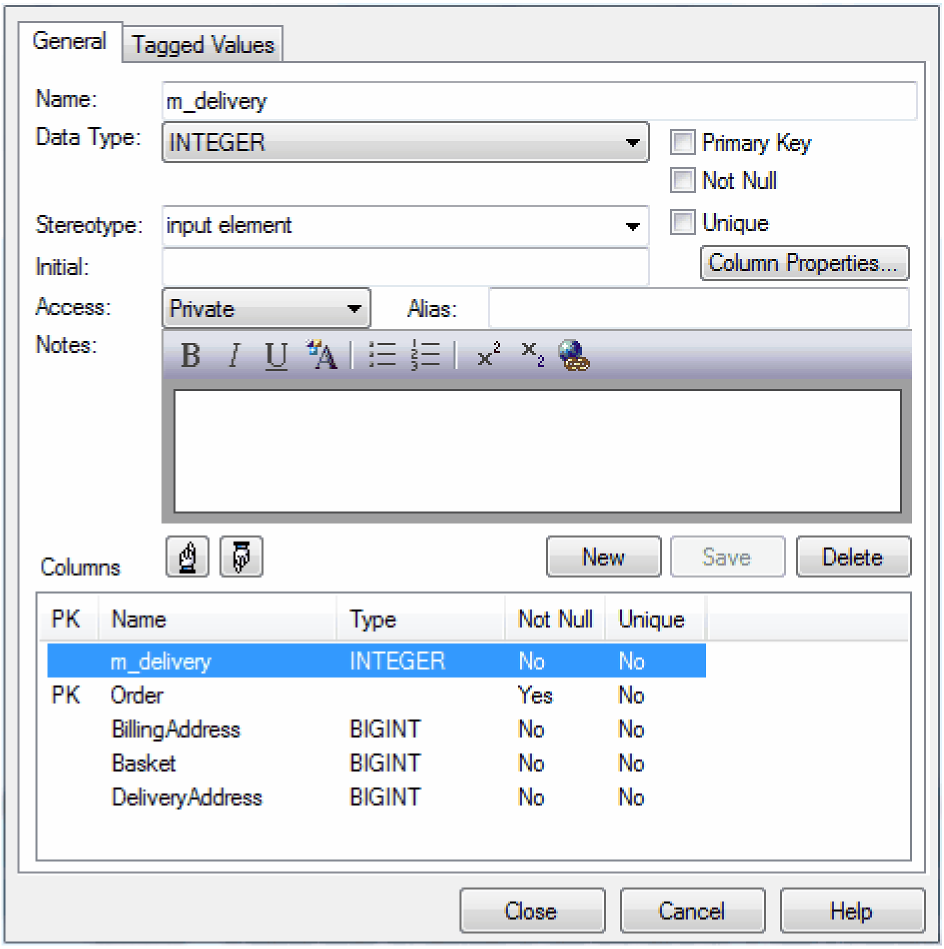
Create Columns

Note:

For MySQL, before creating columns first add ENUM and SET datatypes. Select the **Settings | Database Datatypes** menu option and, on the Database Datatypes dialog, in the **Product Name** field select **MySQL**. Add the datatypes ENUM and SET.

To create columns, follow the steps below:

1. Right-click on the Table in a diagram to open the context menu, and select the **Attributes** menu option.
2. The <Tablename> Columns dialog displays.



Name: m_delivery
Data Type: INTEGER
Primary Key: ☐
Not Null: ☐
Unique: ☐
Stereotype: input element
Initial:
Access: Private
Alias:
Notes:

Columns

PK	Name	Type	Not Null	Unique
	m_delivery	INTEGER	No	No
PK	Order		Yes	No
	BillingAddress	BIGINT	No	No
	Basket	BIGINT	No	No
	DeliveryAddress	BIGINT	No	No

3. In the **Name** field, type the column name.
4. In the **Data Type** field, click on the drop-down arrow and select the data type, and click on the **Save** button.

Tip:

If the drop-down list of datatypes is empty, this means that you have not selected a target database for the table. Close the Columns dialog and re-open the Table Properties dialog to set a database type before continuing. To prevent this recurring, set the default database type 34.

5. The following fields for each column are optional:
   - **Primary Key** - select the checkbox if the column represents the primary key 134 for this table
   - **Not Null** - select the checkbox if empty values are forbidden for this column
   - **Unique** - select the checkbox if it is forbidden for any two values of this column to be identical
   - **Initial** - type a value that can be used as a default value for this column, if required
   - **Access** - click on the drop-down arrow and select a scope of **Private**, **Protected** or **Public** (the field defaults to **Public**)
   - **Alias** - type an alternative name for the field (for display purposes), if any
   - **Notes** - type any other information necessary to document the column; you can format the text using the Rich Text Notes toolbar at the top of the field.

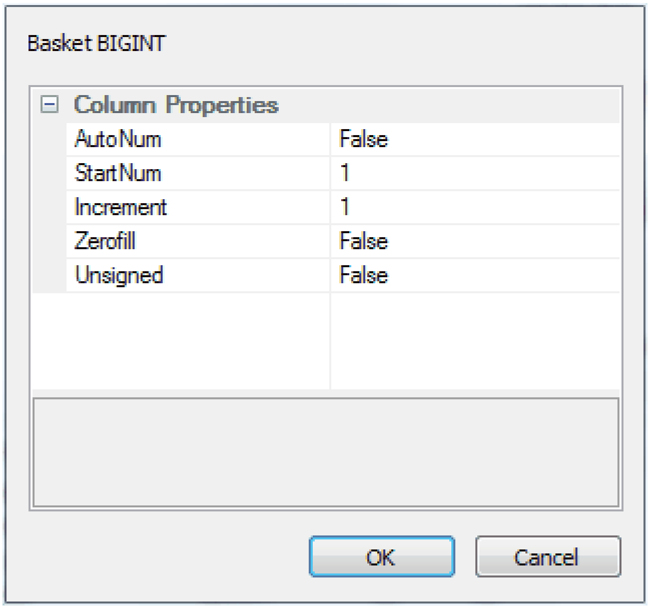
Notes:

- The **unique** characteristic applied to a single column ensures that no two data values in the column can be identical. The **unique** stereotype applied to an index 147 ensures that no two *combinations* of values across a *set* of columns can be identical.
- Some datatypes, such as the Oracle NUMBER type, require a precision and scale. These fields are displayed where required and should be filled in as appropriate. For example, for Oracle:

  create NUMBER by setting Precision = **0** and Scale = **0**
  create NUMBER(8) by setting Precision = **8** and Scale = **0**
  create NUMBER(8,2) by setting Precision = **8** and Scale = **2**.

- Oracle VARCHAR2(15 CHAR) and VARCHAR2(50 BYTE) datatypes can be created by adding the tag *LengthType* with the value **CHAR** or **BYTE**.
- For MySQL ENUM and SET datatypes, in the **Initial** field type the values as a comma-separated list, in the format *('one','two','three'*) or, if one value is the default, in the format: *('one','two','three') default 'three'*.

5. Click on the **Column Properties** button. The Database Columns Properties dialog displays.



Basket BIGINT

Column Properties	
AutoNum	False
StartNum	1
Increment	1
Zerofill	False
Unsigned	False

If you require a sequence, such as an Oracle sequence, select the **AutoNum** property, set the value to **True** and, if necessary, define the start number and increment. Click on the **OK** button to return to the <Tablename> Columns dialog.

6. Click on the **Save** button and on either the **New** button to define another column or the **Close** button to exit from the dialog.

Change the Column Order

To change the column order, follow the steps below:

1. On the Columns dialog, highlight a column name in the Columns panel.
2. Click on the:
   - button to move the column up one position
   - button to move the column down one position.

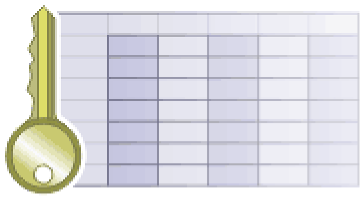
3.5 Create Oracle Packages

To create an Oracle package, follow the steps below:

1. Open the project in the Project Browser and create an Enterprise Architect package (and, if required, a Class diagram).
2. Add a Class element to either the package or the diagram.
3. Open the Properties dialog for the element and, in the **Stereotype** field, type the value **Package**.
4. For the package specification, create an Operation with the name *Specification* and with no return type.
5. Open the Properties dialog for the *Specification* Operation and, on the Behavior tab, type the entire package specification into the **Initial Code** field.
6. For the package body, create an Operation with the name *Body* and with no return type.
7. Open the Properties dialog for the *Body* Operation and, on the Behavior tab, type the entire package body into the **Initial Code** field.

For information on the objects mentioned above, see the *Work With Elements* section of *UML Modeling With Enterprise Architect - UML Modeling Tool.*

3.6 Primary Key



What is a Primary Key?

Keys are used to access tables, and come in two varieties: Primary Keys and Foreign Keys. A Primary Key uniquely identifies a record in a table, while a Foreign Key 137 accesses data in some other related table via its Primary Key.

Define a Simple Primary Key

If a Primary Key consists of a single column, it is very easy to define.

1. Right-click on the table in a diagram to display the context menu. Select the **Attributes** menu option.
2. In the Attributes dialog, select the column that makes up the Primary Key.
3. Select the **Primary Key** checkbox and click on the **Save** button.

A stereotyped operation is automatically created. It is this operation that defines the Primary Key for the table. To remove a Primary Key, simply delete this operation.

Define a Complex Primary Key

Often, a Primary Key consists of more than one column. For example, a column *LastName* might not be unique within a table, so a Primary Key is created from the *LastName*, *FirstName* and *DateOfBirth* columns. Perform the following steps to create a complex Primary Key:

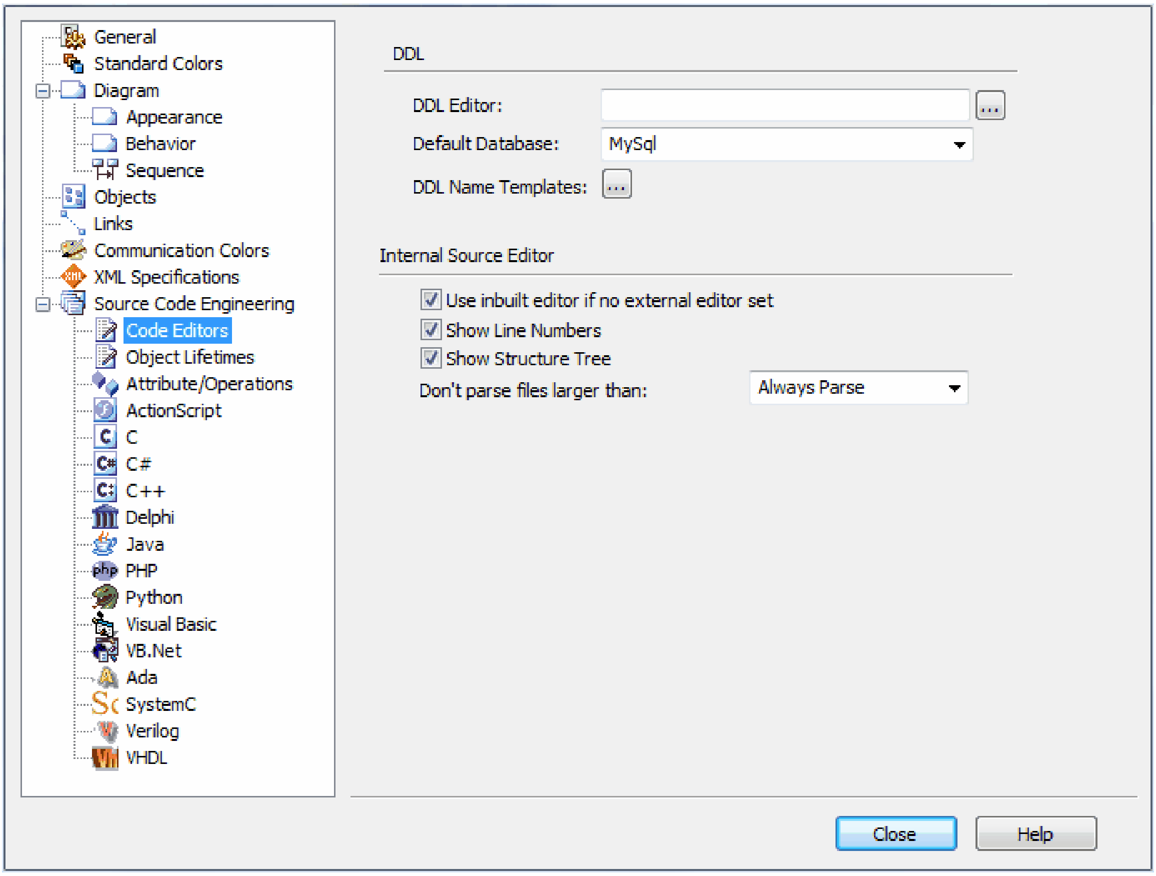
1. Follow the steps above to create a Simple Primary Key. It doesn't matter which column you choose.
2. Right-click on the table in a diagram to open the context menu. Select the **Operations** menu option.
3. Select the Primary Key operation (its name begins with **PK_**) and then click on the Column tab.
4. To add a column to the Primary Key, click on the **New** button, select a column from the Column Name list box, and then click on the **Save** button.
5. Click on the **Hand** buttons (up and down arrow) to change the order of columns in the Primary Key, if necessary.

(See also the *SQL Server Non-Clustered Primary Keys* 136 topic).

Define a Primary Key Name Template

To define the name template for a Primary Key, follow the steps below:

1. Select the **Tools | Options | Source Code Engineering | Code Editors** menu option. The DDL page of the Options dialog displays.



DDL

DDL Editor:

Default Database: MySql

DDL Name Templates: ...

Internal Source Editor

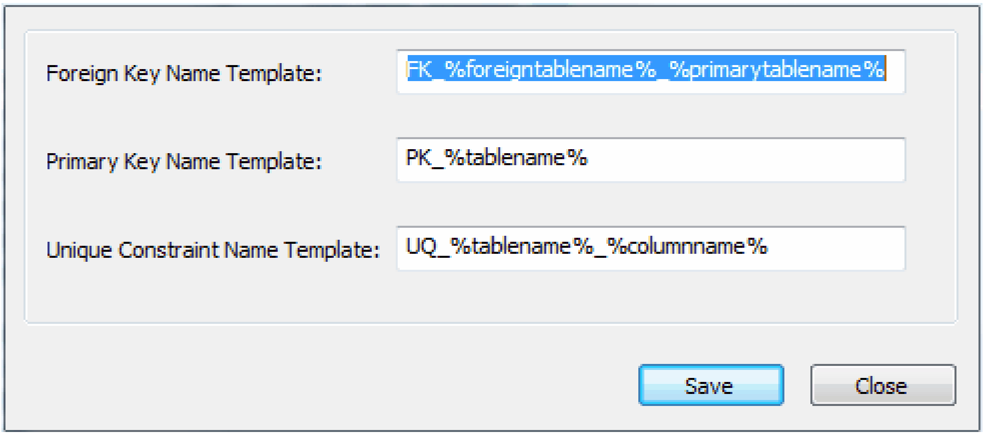
☑ Use inbuilt editor if no external editor set

☑ Show Line Numbers

☑ Show Structure Tree

Don't parse files larger than: Always Parse

2. Click on the **DDL Name Templates** button. The DDL Name Template dialog displays, showing the default name templates.



Foreign Key Name Template: FK_%foreigntablename%_%primarytablename%

Primary Key Name Template: PK_%tablename%

Unique Constraint Name Template: UQ_%tablename%_%columnname%

3. Edit or replace the template in the **Primary Key Name Template** field.

Note:

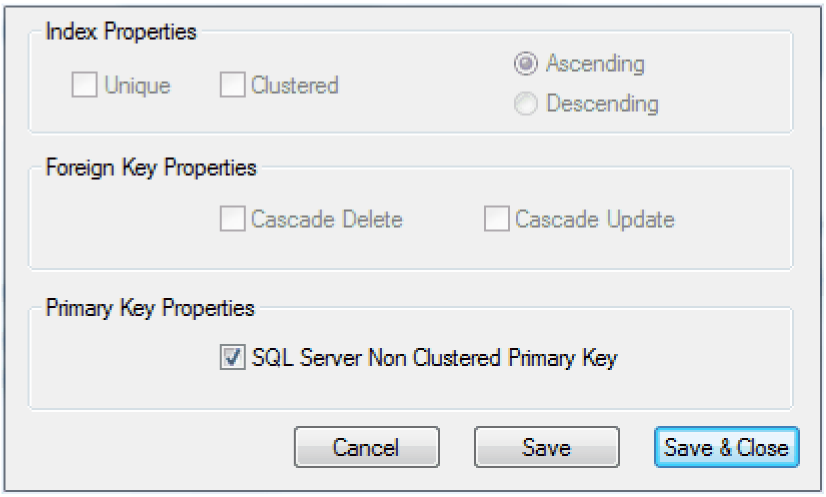
If you want to display the Primary Key description as *PK_tablename_columnname* then change the **Primary Key Name Template** field to *PK_%tablename%_%columnname%.*

4. Click on the **Save** button.

3.6.1 SQL Server Non Clustered Keys

To define a primary key as non-clustered for a SQL Server table, follow the steps below:

1. Right-click on the table in a diagram to open the context menu.
2. Select the **Operations** menu option. The Table Operations dialog displays.
3. Highlight the Primary Key Operation and select **Extended Properties**. The Database Operation Properties dialog displays.



Index Properties

☐ Unique ☐ Clustered ● Ascending ○ Descending

Foreign Key Properties

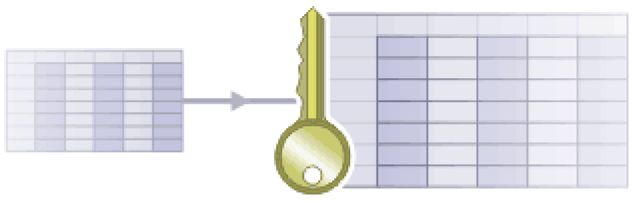
☐ Cascade Delete ☐ Cascade Update

Primary Key Properties

☑ SQL Server Non Clustered Primary Key

4. Select the **SQL Server Non Clustered Primary Key** checkbox.
5. Click on the **Save & Close** button.

3.7 Foreign Key



What is a Foreign Key?

Two types of key are used to access tables: Primary Keys 134 and Foreign Keys. A Primary Key uniquely identifies a record in a table, while a Foreign Key accesses data in some other related table via its Primary Key.

Foreign keys are represented in Enterprise Architect UML using stereotyped operations. A Foreign Key is a collection of columns (attributes) that together have some operational meaning (they enforce a relationship to a Primary Key in another table). A Foreign Key is modeled as an operation stereotyped with the *FK* stereotype; the operation parameters become the columns involved in the key.

Note:

It isn't necessary to define a Foreign Key in order to access another table through its Primary Key. Foreign Keys are a feature of some database management systems, providing 'extras' such as referential integrity checking that prevents the deletion of a record if its Primary Key value exists in some other table's Foreign Key. The same thing can be achieved programmatically.

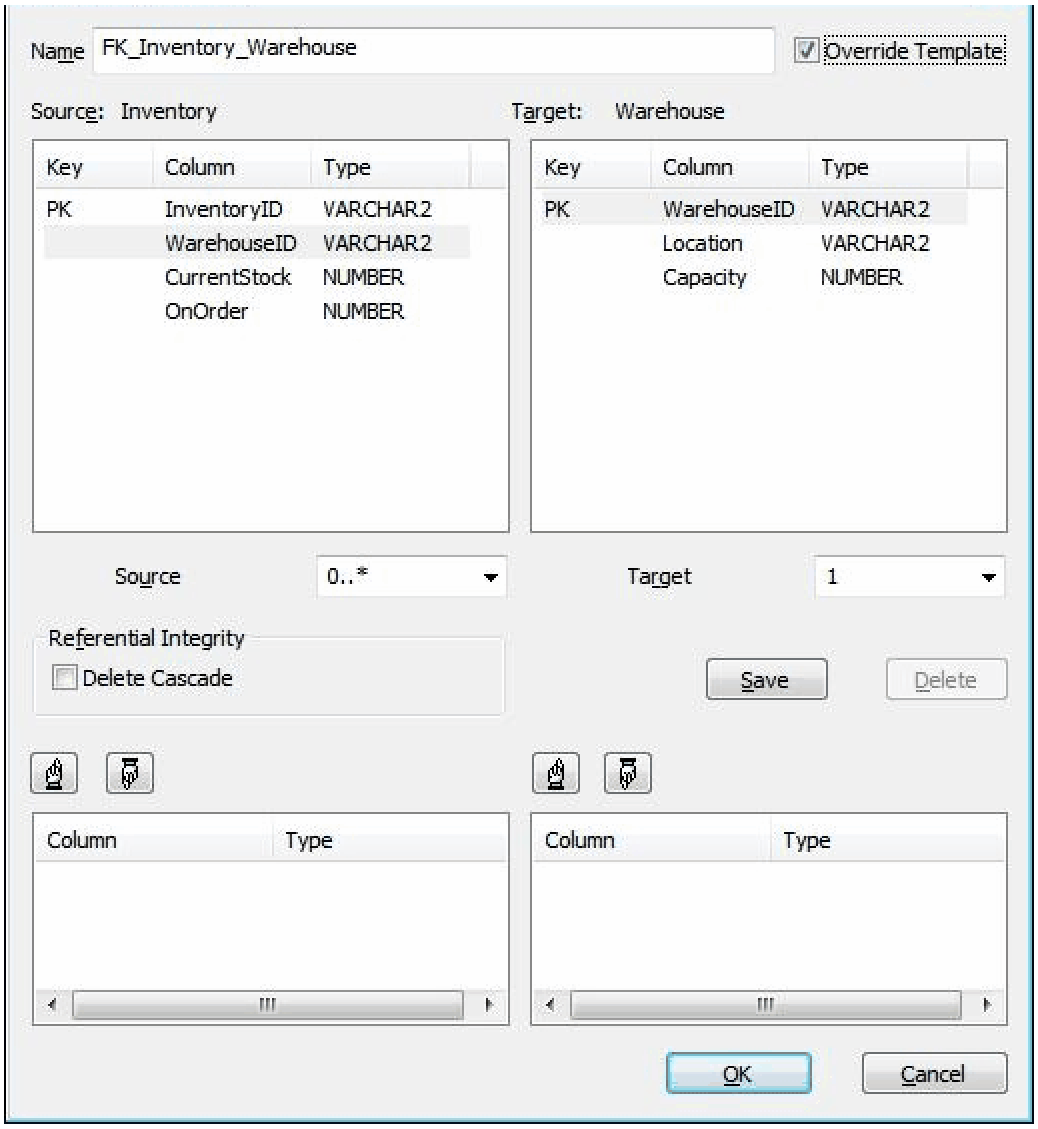
To create a Foreign Key 137, click on the link.

You might also have to define a Name Template 141 for Foreign Keys.

3.7.1 Create Foreign Key

To create a Foreign Key, follow the steps below:

1. Locate the required Tables in a diagram. Both tables must have defined database types 125.
2. Select an *Associate* connector in the Class Relationships page of the Enterprise Architect UML Toolbox .
3. Click on the Table to contain the Foreign Key (source) and draw the connector to the other Table (target).
4. Right-click on the connector to display the context menu, and select the **Foreign Keys** option. The Foreign Key Constraint dialog displays.



Name: FK_Inventory_Warehouse
[✓] Override Template

Source: Inventory

Key	Column	Type
PK	InventoryID	VARCHAR2
	WarehouseID	VARCHAR2
	CurrentStock	NUMBER
	OnOrder	NUMBER

Target: Warehouse

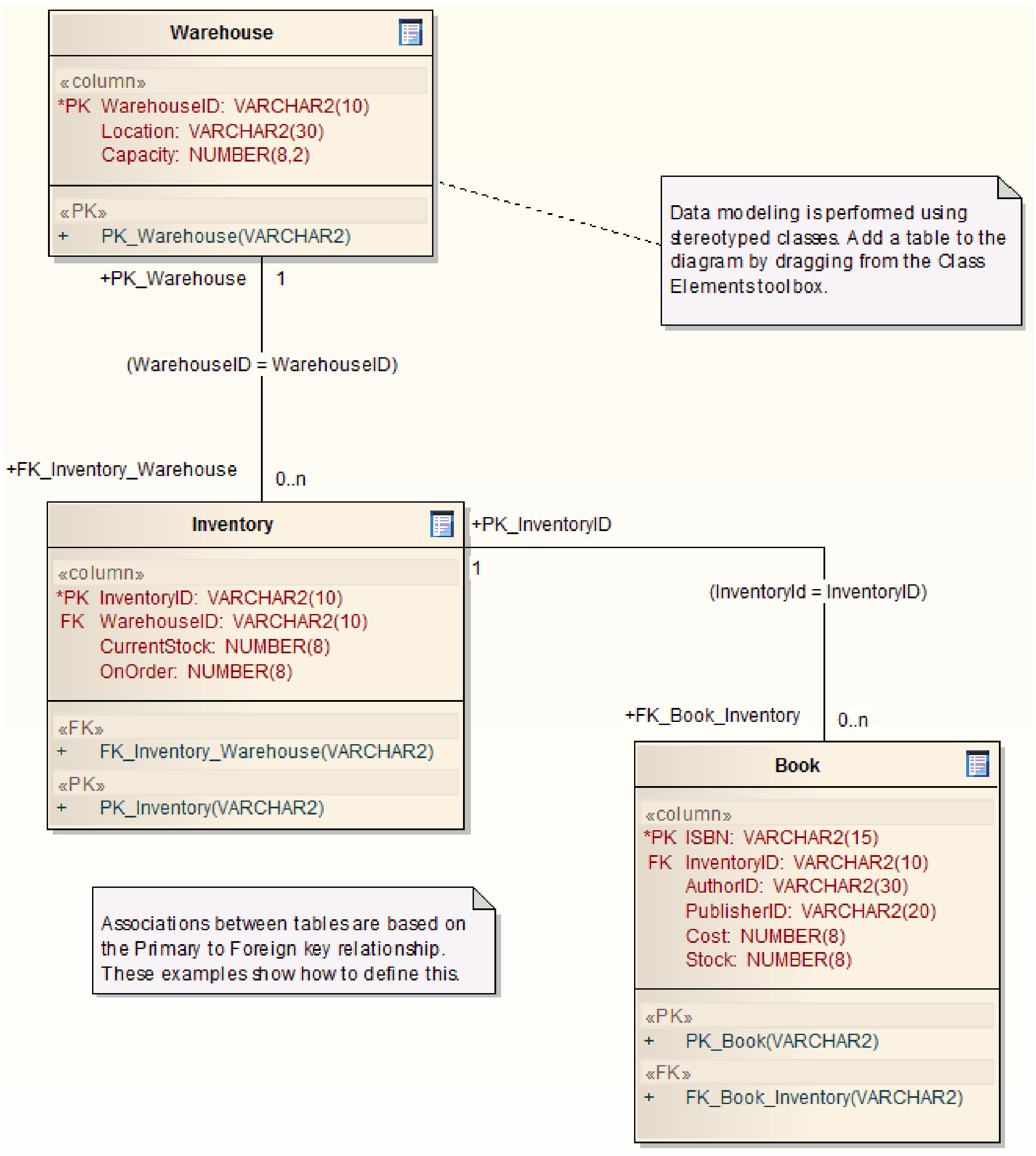
Key	Column	Type
PK	WarehouseID	VARCHAR2
	Location	VARCHAR2
	Capacity	NUMBER

Source: 0..*
Target: 1

Referential Integrity
[ ] Delete Cascade

5. The default foreign key name is set by the Foreign Key Name Template. To change the name to something other than the default provided by the template, select the **Override Template** checkbox and edit the foreign key name.
6. Highlight the columns involved in the Foreign Key relationship.
7. Click on the **Save** button to automatically generate the Foreign Key operations.

You have created the Foreign Key. The example below shows how this looks in a diagram:


Warehouse
«column»
*PK WarehouseID: VARCHAR2(10)
Location: VARCHAR2(30)
Capacity: NUMBER(8,2)
«PK»
+ PK_Warehouse(VARCHAR2)
+PK_Warehouse
1
(WarehouseID = WarehouseID)
+FK_Inventory_Warehouse
0..n
Data modeling is performed using stereotyped classes. Add a table to the diagram by dragging from the Class Elements toolbox.
Inventory
«column»
*PK InventoryID: VARCHAR2(10)
FK WarehouseID: VARCHAR2(10)
CurrentStock: NUMBER(8)
OnOrder: NUMBER(8)
«FK»
+ FK_Inventory_Warehouse(VARCHAR2)
«PK»
+ PK_Inventory(VARCHAR2)
+PK_InventoryID
1
(InventoryId = InventoryID)
+FK_Book_Inventory
0..n
Book
«column»
*PK ISBN: VARCHAR2(15)
FK InventoryID: VARCHAR2(10)
AuthorID: VARCHAR2(30)
PublisherID: VARCHAR2(20)
Cost: NUMBER(8)
Stock: NUMBER(8)
«PK»
+ PK_Book(VARCHAR2)
«FK»
+ FK_Book_Inventory(VARCHAR2)
Associations between tables are based on the Primary to Foreign key relationship. These examples show how to define this.

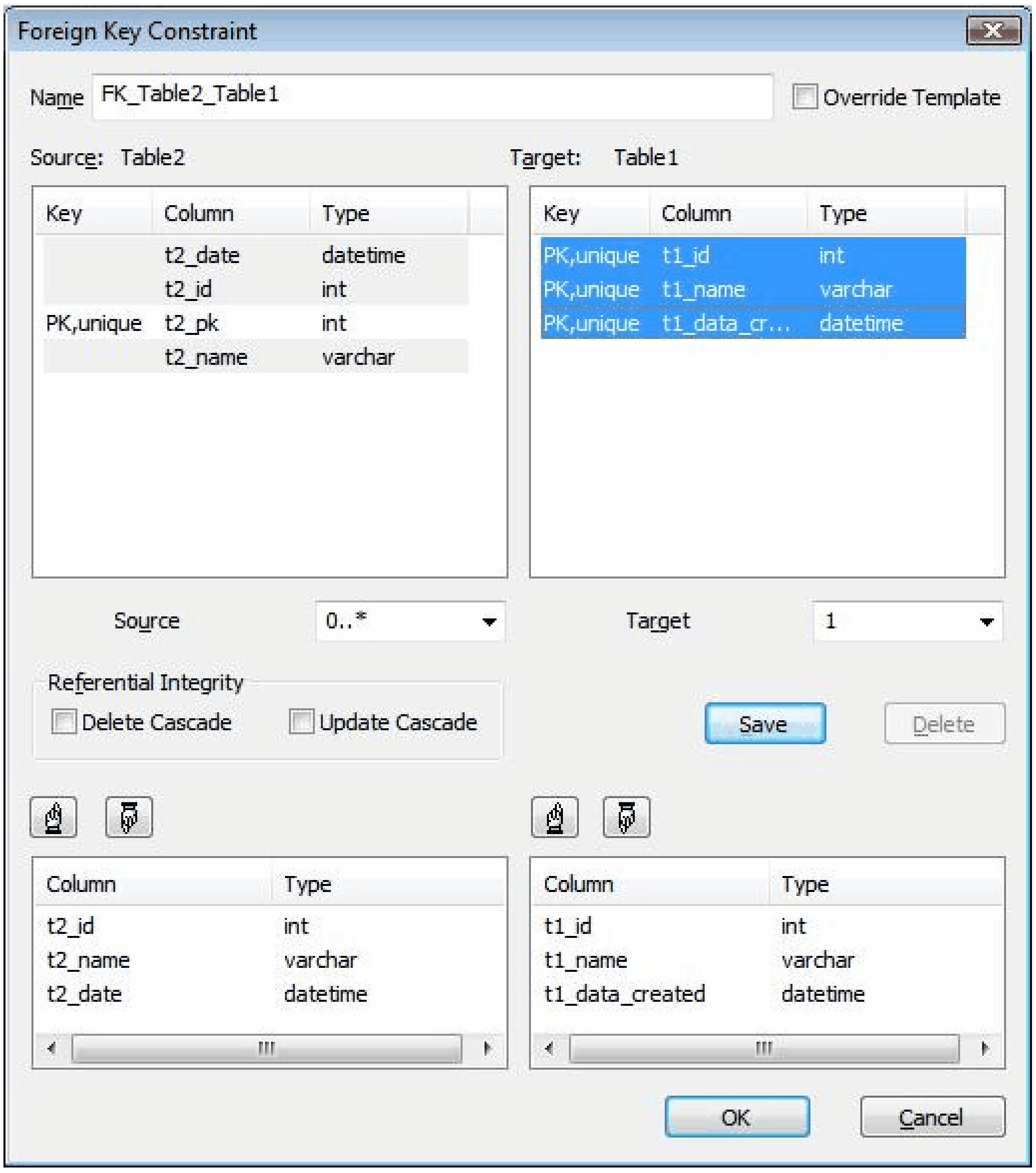
Composite Foreign Key

To create a composite Foreign Key, select the appropriate columns and click on the **Save** button. The Foreign Key columns are sorted according to datatype to match the datatypes of the targeted composite Primary Key.

If required, you can change the order of the key columns by clicking on the and buttons.

Tip:

If you are defining a MySQL database and want to use Foreign Keys, you must set the table type 126 to enable this.



Name: FK_Table2_Table1

Override Template: [ ]

Source: Table2

Key	Column	Type
	t2_date	datetime
	t2_id	int
PK,unique	t2_pk	int
	t2_name	varchar

Target: Table1

Key	Column	Type
PK,unique	t1_id	int
PK,unique	t1_name	varchar
PK,unique	t1_data_cr...	datetime

Source: 0..*

Target: 1

Referential Integrity

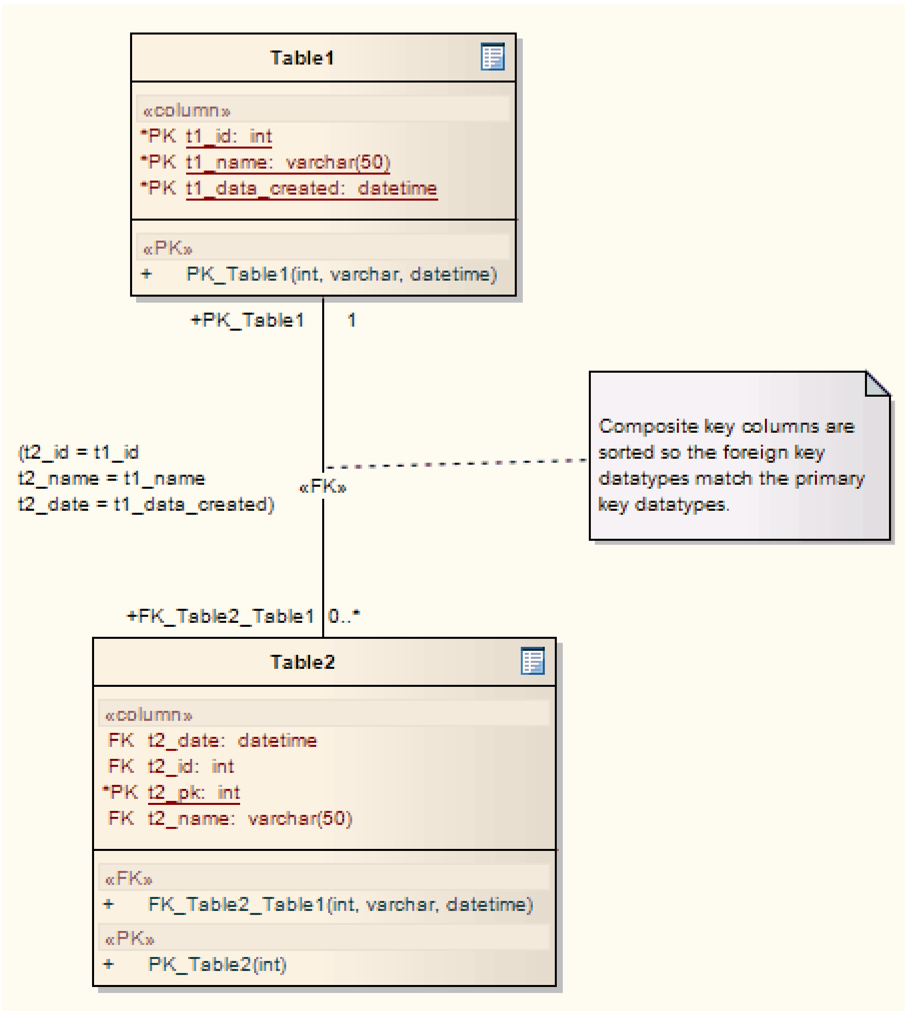
Delete Cascade: [ ]

Update Cascade: [ ]

Column	Type
t2_id	int
t2_name	varchar
t2_date	datetime

Column	Type
t1_id	int
t1_name	varchar
t1_data_created	datetime

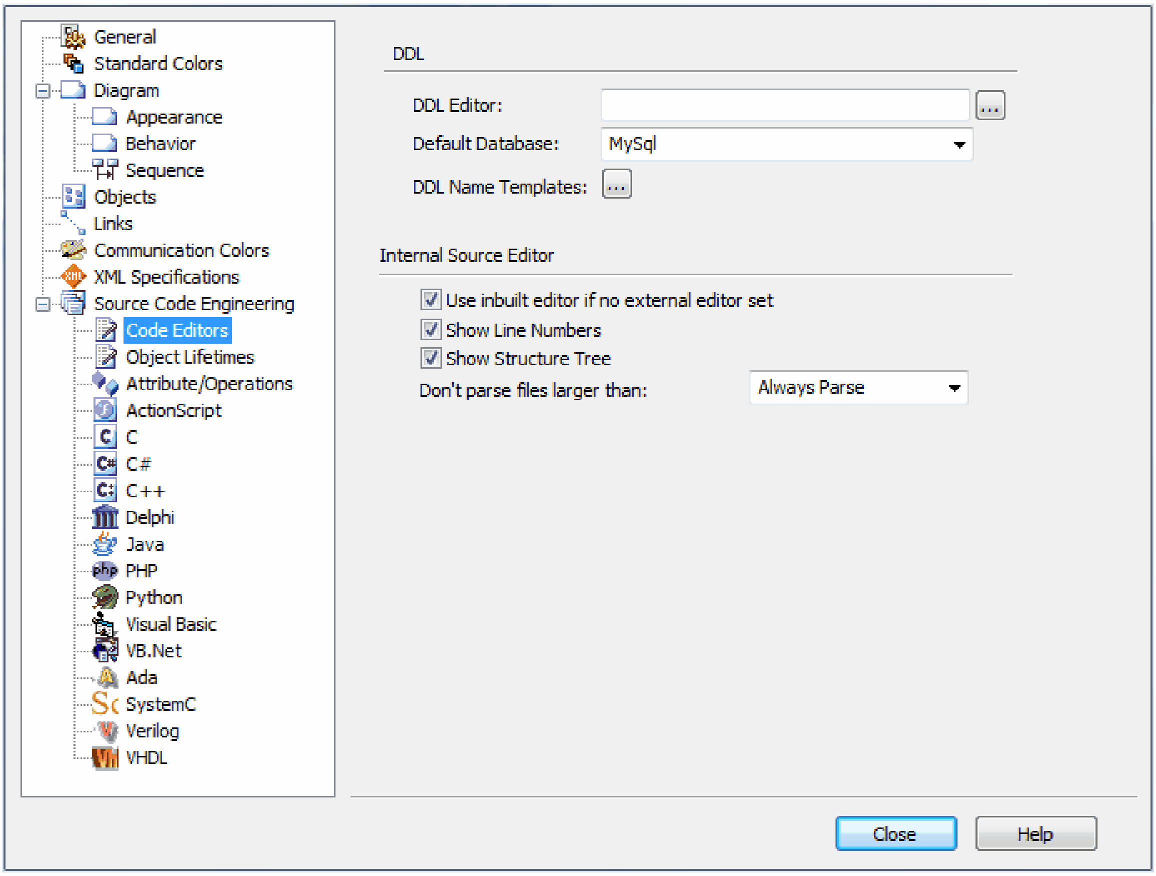
This creates the composite Foreign Key. The example below shows how this looks in a diagram:



3.7.2 Define Foreign Key Name Template

To define the name template for a Foreign Key, follow the steps below:

1. Select the **Tools | Options | Source Code Engineering | Code Editors** menu option. The DDL page of the Options dialog displays.



DDL

DDL Editor:

Default Database: MySql

DDL Name Templates: ...

Internal Source Editor

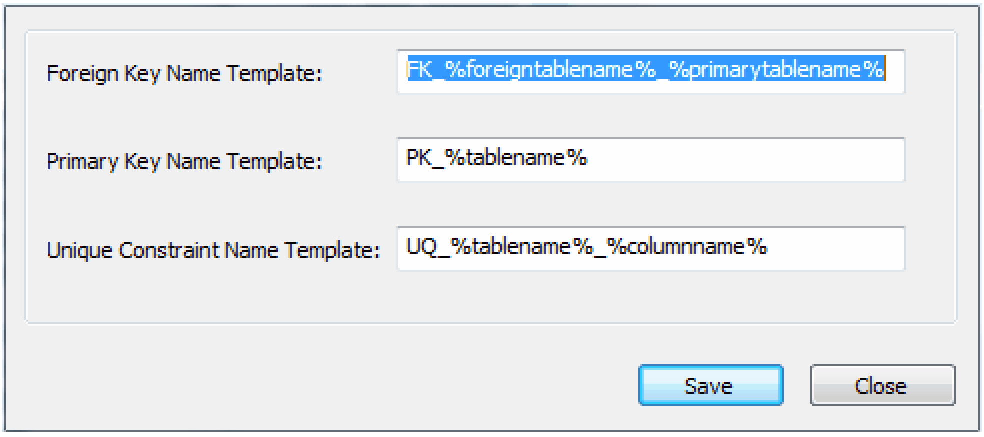
☑ Use inbuilt editor if no external editor set

☑ Show Line Numbers

☑ Show Structure Tree

Don't parse files larger than: Always Parse

2. Click on the **DDL Name Template** button. The DDL Name Template dialog displays, showing the default name templates.



Foreign Key Name Template: FK_%foreigntablename%_%primarytablename%

Primary Key Name Template: PK_%tablename%

Unique Constraint Name Template: UQ_%tablename%_%columnname%

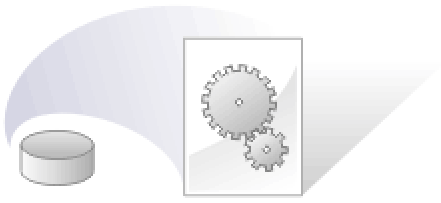
3. Edit or replace the name template in the **Foreign Key Name Template** field.

Note:

If you want to display the Foreign Key description as *FK_foreigntablename_FKcolumnname_primarytablename_PKcolumnname* then change the **Foreign Key Name Template** field to *FK_%foreigntablename%_%fkcolumnname%_%primarytablename%_%pkcolumnname%.*

4. Click on the **Save** button.

3.8 Stored Procedures



What is a Stored Procedure?

A stored procedure is a group of SQL statements that form a logical unit and perform a particular task. Stored procedures are used to encapsulate a set of operations or queries to execute on a database server. You can compile and execute stored procedures with different parameters and results, and they can have any combination of input, output and input/output parameters.

Enterprise Architect models stored procedures as individual Classes 143.

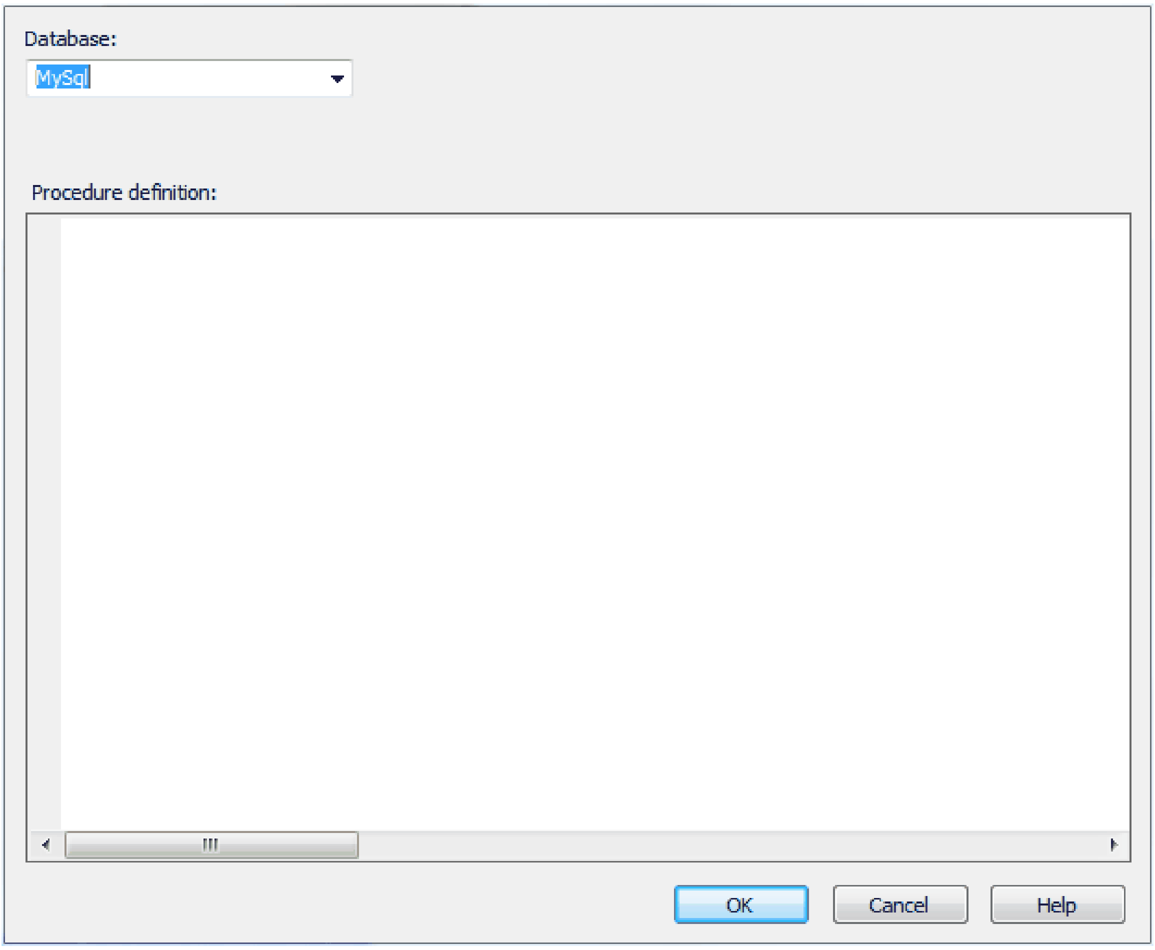
Note:

Stored procedures are currently supported for: DB2; SQL Server; Firebird/Interbase; Informix; Ingres; Oracle 9i, 10g and 11g; MySQL; PostgreSQL; Sybase Adaptive Server Enterprise (ASE) and Sybase Adaptive Server Anywhere (ASA).

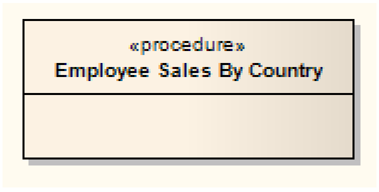
3.8.1 Create Individual Class Procedure

To create a stored procedure as an individual Class, follow the steps below:

1. Open the required diagram.
2. From the Data Modeling page of the Enterprise Architect UML Toolbox (**More tools | Data Modeling**) drag the **Procedure** icon onto the diagram.
3. If the Properties dialog does not automatically display, double-click on the element.



4. In the **Database** field click on the drop-down arrow and select the target DBMS to model. (The field displays the default database if it has already been set.)
5. In the **Procedure definition** field, type the entire procedure text.
6. Click on the **OK** button.



To define a name for the stored procedure, click on the element, click on the name (**Class<n>**) and click again. This highlights the text for editing. Type in the required name. (For further details, see the *In-Place Editing* section in *UML Modeling With Enterprise Architect - UML Modeling Tool.*)

3.9 Views

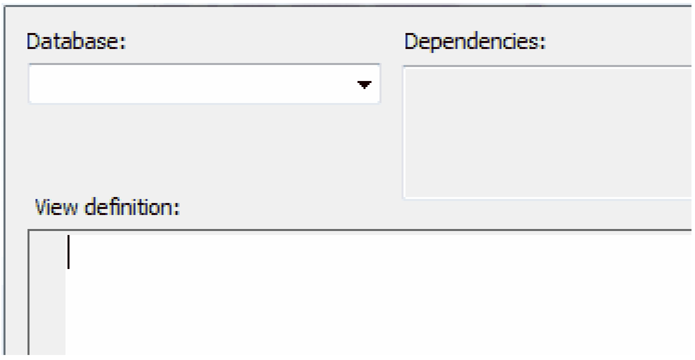
Note:

Views are currently supported for: DB2; SQL Server; Firebird/Interbase; Informix; Ingres; Oracle 9i, 10g and 11g; MySQL; PostgreSQL; Sybase Adaptive Server Enterprise (ASE) and Sybase Adaptive Server Anywhere (ASA).

Create a View

To create a database View, follow the steps below:

1. On the Data Modeling page of the Enterprise Architect UML Toolbox (**More tools | Data Modeling**), drag the **View** icon onto your Data Modeling diagram.
2. If the View Properties dialog does not immediately display, double-click on the element.



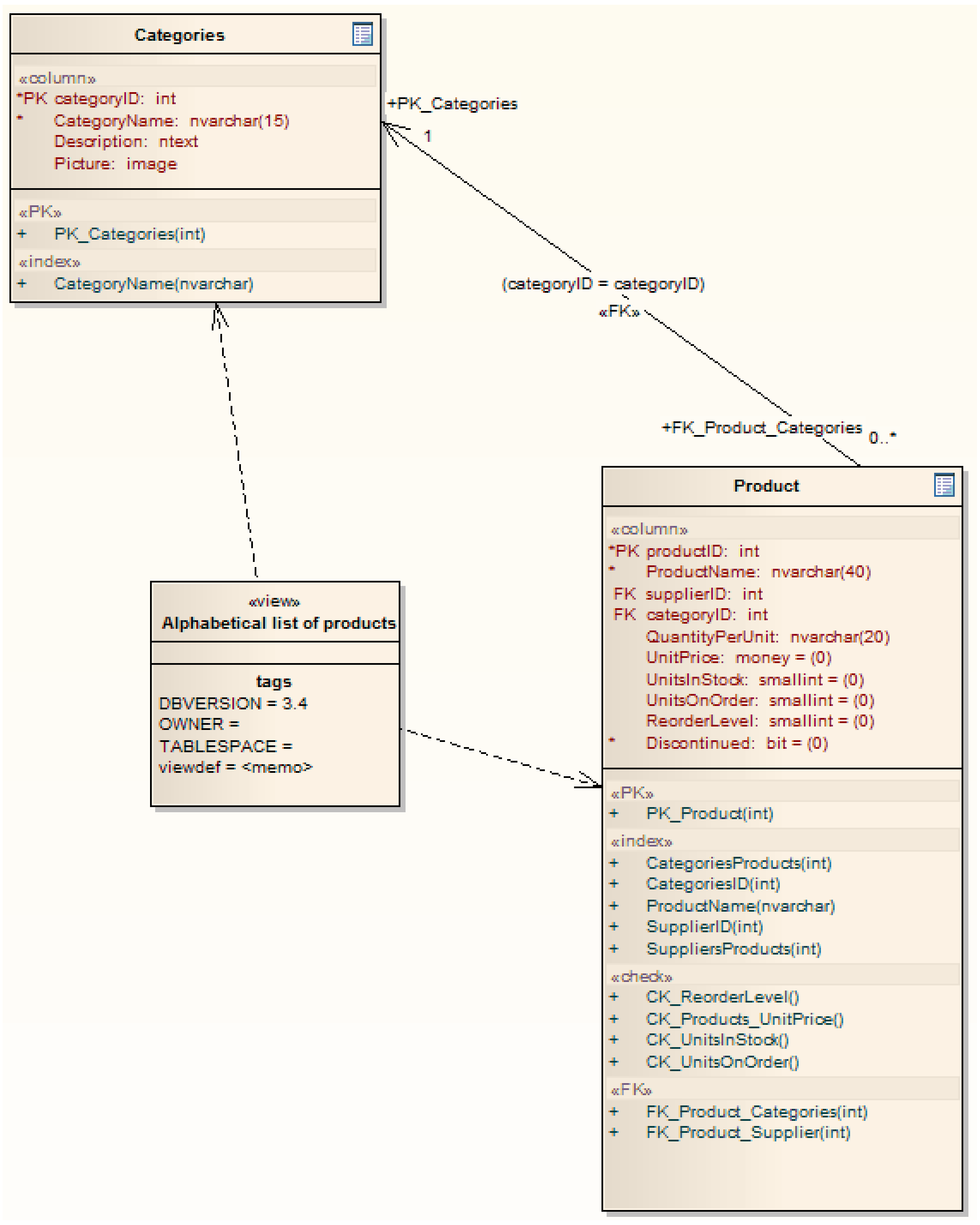
3. From the **Database** drop-down list, select the target DBMS to model. The default database displays if it has already been set.
4. Click on the **OK** button.

To define a name for the View, click on the element, click on the name (**Class<n>**) and click again. This highlights the text for editing. Type in the required name. (For further details, see the *In-Place Editing* section in *UML Modeling With Enterprise Architect - UML Modeling Tool.*)

Define View Properties

1. Create a Dependency connector from the View to the table or tables on which the View depends.
2. Double-click on the View to display the Properties dialog. The tables are now listed in the **Dependencies** field.
3. In the **View definition** field, type the full view definition. (The code editor provides intellisense for basic SQL keywords and functions - see *Using Enterprise Architect - UML Modeling Tool*).
4. Click on the **OK** button to save your definition.

The View definition and certain other parameters are held as Tagged Values. The View definition is held in the *viewdef* memo Tagged Value, as shown in the following example diagram. You can select and view the *viewdef* Tagged Value in the Tagged Values window, and include it in RTF reports by inserting the **valueOf(viewdef)** field in the *Package::Element* or *Element::Tagged Values* sections (see *Report Creation in UML Models*).


Categories
«column»
*PK categoryID: int
* CategoryName: nvarchar(15)
Description: ntext
Picture: image
«PK»
+ PK_Categories(int)
«index»
+ CategoryName(nvarchar)
+PK_Categories
1
(categoryID = categoryID)
«FK»
+FK_Product_Categories
0..*
Product
«column»
*PK productID: int
* ProductName: nvarchar(40)
FK supplierID: int
FK categoryID: int
QuantityPerUnit: nvarchar(20)
UnitPrice: money = (0)
UnitsInStock: smallint = (0)
UnitsOnOrder: smallint = (0)
ReorderLevel: smallint = (0)
* Discontinued: bit = (0)
«PK»
+ PK_Product(int)
«index»
+ CategoriesProducts(int)
+ CategoriesID(int)
+ ProductName(nvarchar)
+ SupplierID(int)
+ SuppliersProducts(int)
«check»
+ CK_ReorderLevel()
+ CK_Products_UnitPrice()
+ CK_UnitsInStock()
+ CK_UnitsOnOrder()
«FK»
+ FK_Product_Categories(int)
+ FK_Product_Supplier(int)
«view»
Alphabetical list of products
tags
DBVERSION = 3.4
OWNER =
TABLESPACE =
viewdef = <memo>

3.10 Index, Trigger, Check Constraint

What is an Index?

An index is a sorted look-up for a table. When it is known in advance that a table must be sorted in a specific order, it is usually worth the small processing overhead to always maintain a sorted look-up list rather than sort the table every time it is required. In Enterprise Architect, an index is modeled as a stereotyped operation. On generating DDL, the necessary instructions for generating indexes are written to the DDL output.

The **unique** characteristic applied to a single column ensures that no two data values in the column can be identical. The **unique** stereotype applied to an index ensures that no two *combinations* of values across a *set* of columns can be identical.

What is a Trigger?

A trigger is an operation automatically executed as a result of the modification of data in the database, and usually ensures consistent behavior of the database. For example, a trigger might be used to define validations that must be performed every time a value is modified, or might perform deletions in a secondary table when a record in the primary table is deleted. In Enterprise Architect, a trigger is modeled as a stereotyped operation. Currently Enterprise Architect does not generate DDL for triggers, but nonetheless they aid in describing and specifying the table structure in detail.

What is a Check Constraint?

A *Check Constraint* enforces domain integrity by limiting the values that are accepted by a column.

Create an Index

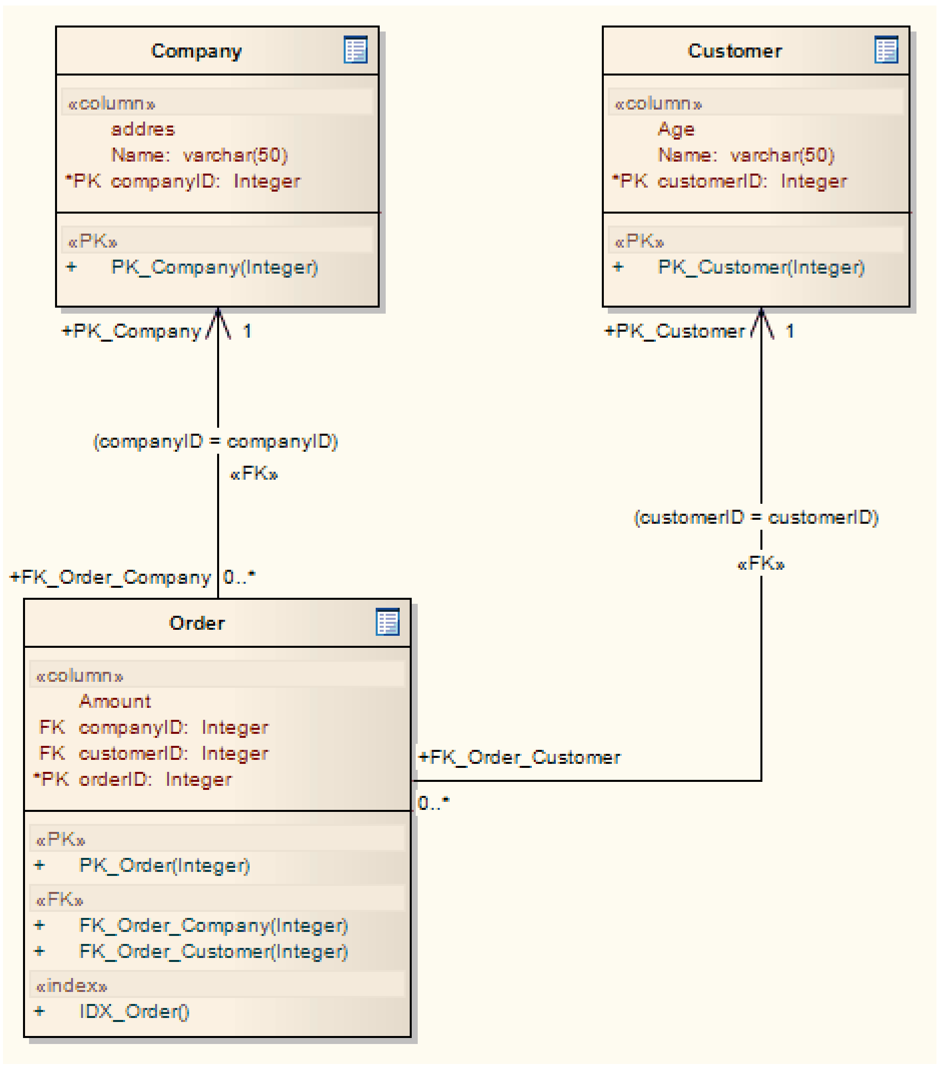
Ensure that the column(s) to be used in the index have already been defined 130 in the table.

1. Right-click on the required table either in a diagram or in the Project Browser.
2. Select the **Operations** context menu option. The Operations dialog displays.
3. Add an operation (with a name such as *IDX_CustomerID*; the *IDX_* prefix is optional but it helps identify the operation).
4. In the **Stereotype** field for the operation, select **index** (**check** and **unique** are also supported).
5. Click on the Column tab.
6. Select the required columns from the **Columns** drop-down list in the required order, then click on the **Save** button to save changes.

Create a Check Constraint or Trigger

1. Locate the required table in either a diagram or the Project Browser.
2. Use the context menu to open the Operations dialog.
3. Add an operation (such as *CHK_ColumnName* or *TRG_OnCustomerUpdate*; the *CHK_* and *TRG_* prefixes are optional but help identify the operation).
4. In the **Stereotype** field for the constraint, select **check** or **trigger** as appropriate and click on the **Save** button to save changes.
5. Select the constraint operation, then the Behavior tab.
6. Enter the entire check constraint clause (for example, **col1 < 1000**), or the entire trigger code (including the *CREATE_TRIGGER* statement) in the **Initial Code** field and click on the **Save** button to save changes.

The example below shows how an index looks in a diagram (in the *Order* element):


Company
«column»
addres
Name: varchar(50)
*PK companyID: Integer
«PK»
+ PK_Company(Integer)
Customer
«column»
Age
Name: varchar(50)
*PK customerID: Integer
«PK»
+ PK_Customer(Integer)
+PK_Company
1
(companyID = companyID)
«FK»
+FK_Order_Company
0..*
+PK_Customer
1
(customerID = customerID)
«FK»
+FK_Order_Customer
0..*
Order
«column»
Amount
FK companyID: Integer
FK customerID: Integer
*PK orderID: Integer
«PK»
+ PK_Order(Integer)
«FK»
+ FK_Order_Company(Integer)
+ FK_Order_Customer(Integer)
«index»
+ IDX_Order()

3.11 Generate DDL For a Table

Note:

In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Generate Source Code and DDL** permission to generate DDL. See *User Security in UML Models.*

To generate simple DDL scripts to create the tables in your model, follow the steps below:

1. In the diagram, right-click on the table for which to generate DDL. The context menu displays.
2. Select the **Generate DDL** option. The Generate DDL dialog displays.

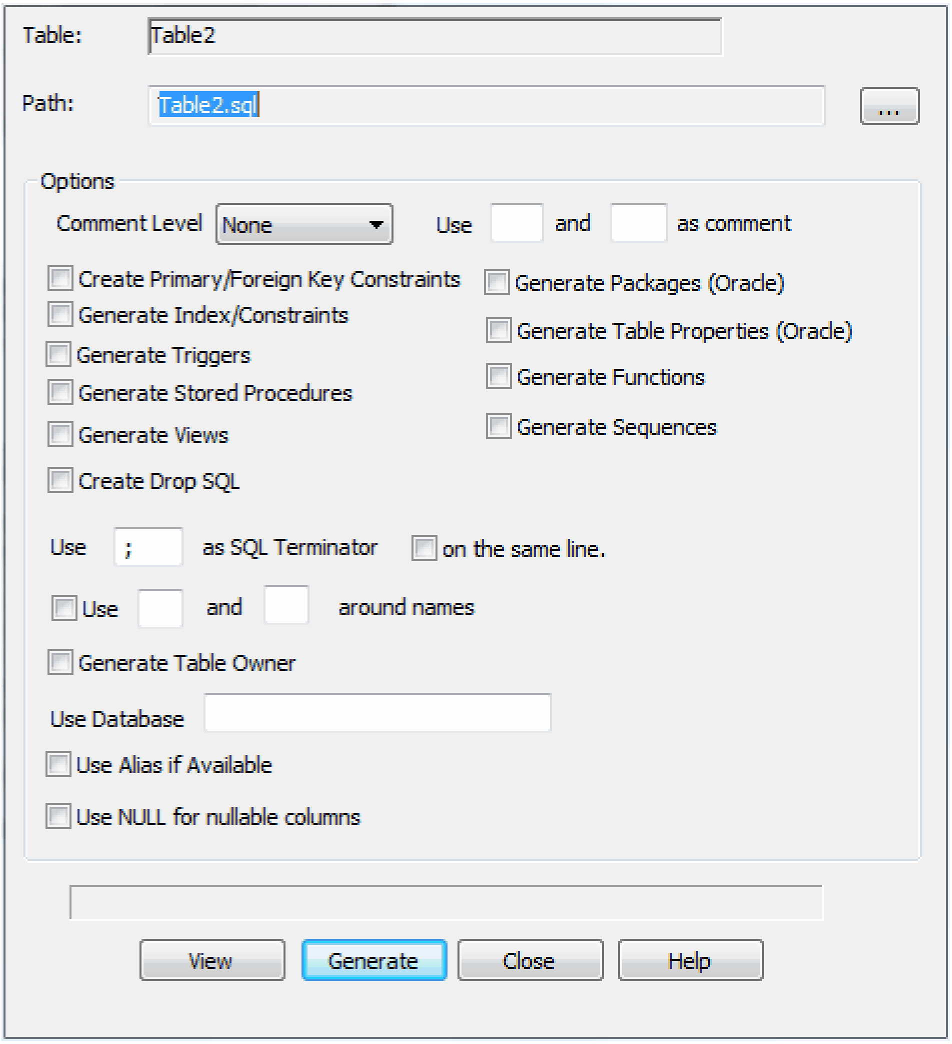


Table: Table2

Path: Table2.sql

Options

Comment Level: None Use [ ] and [ ] as comment

[ ] Create Primary/Foreign Key Constraints
[ ] Generate Index/Constraints
[ ] Generate Triggers
[ ] Generate Stored Procedures
[ ] Generate Views
[ ] Create Drop SQL
[ ] Generate Packages (Oracle)
[ ] Generate Table Properties (Oracle)
[ ] Generate Functions
[ ] Generate Sequences

Use ; as SQL Terminator [ ] on the same line.

[ ] Use [ ] and [ ] around names

[ ] Generate Table Owner

Use Database

[ ] Use Alias if Available

[ ] Use NULL for nullable columns

3. In the **Path** field, use the **[...]** (Browse) button to select the filename of the script to create.
4. To include comments in the DDL, in the **Comment Level** field select the appropriate level. For example, **Column** for comments on columns, or **All** for comments on all structures.
5. Select the checkboxes for the appropriate inclusions. For example, to include a 'drop table' command in the script, select the **Create Drop SQL** checkbox. Deselect the checkboxes for inclusions you do not require.

Notes:

- Some checkboxes display only if the appropriate database is defined for the table. For example, **IF EXISTS** displays only if the database for the table is PostgreSQL.
- If generating Oracle sequences, you must always select the **Generate Triggers** and **Generate Sequences** checkboxes; this ensures that a pre-insert trigger is generated to select the next sequence value to populate the column. Also select the **Auto Numbering** 130 checkbox 130 in the column properties.

6. To create the DDL, click on the **Generate** button.
7. To view the output, click on the **View** button (you must configure a DDL viewer in the Local Settings dialog first).

Note:

You can transport these DDL scripts between models, using the **Export Reference Data** and **Import Reference Data** options on the **Tools** menu. See the *Reference Data* topic in *UML Model Management.*

3.12 Generate DDL for a Package

Note:

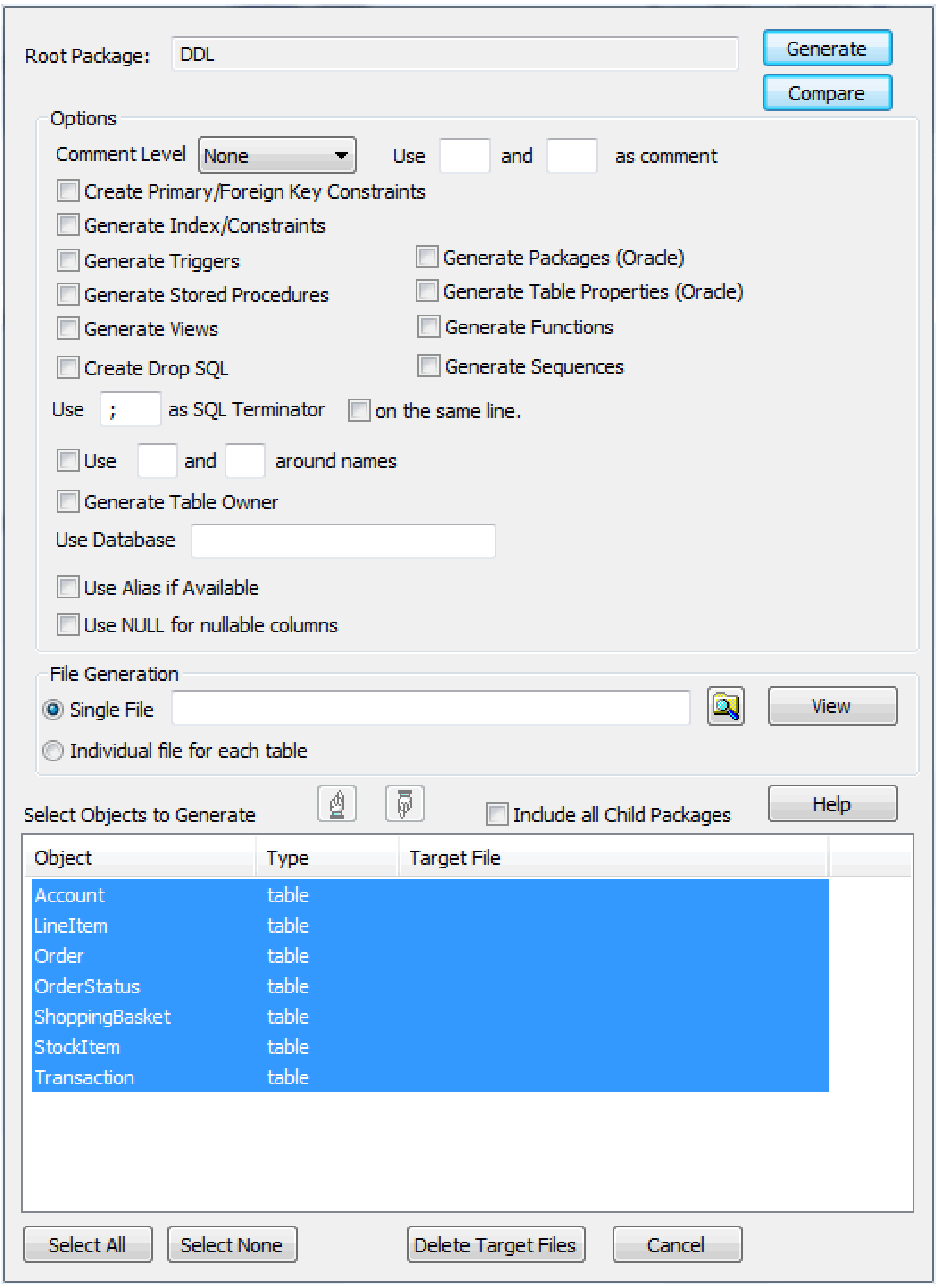
In the Corporate, Business and Software Engineering, System Engineering and Ultimate editions of Enterprise Architect, if security is enabled you must have **Generate Source Code and DDL** permission to generate DDL. See *User Security in UML Models.*

In this procedure, you can generate DDL for a package, and also compare the DDL with the database.

Generate DDL

To generate DDL for a package, follow the steps below:

1. Right-click on the required package in the Project Browser. The context menu displays.
2. Select the **Code Engineering | Generate DDL** menu option. The Generate Package DDL dialog displays.



Root Package: DDL

Options

Comment Level None Use and as comment

- [ ] Create Primary/Foreign Key Constraints
- [ ] Generate Index/Constraints
- [ ] Generate Triggers
- [ ] Generate Stored Procedures
- [ ] Generate Views
- [ ] Create Drop SQL
- [ ] Generate Packages (Oracle)
- [ ] Generate Table Properties (Oracle)
- [ ] Generate Functions
- [ ] Generate Sequences

Use ; as SQL Terminator [ ] on the same line.

[ ] Use and around names

- [ ] Generate Table Owner

Use Database

- [ ] Use Alias if Available
- [ ] Use NULL for nullable columns

File Generation

● Single File

○ Individual file for each table

Select Objects to Generate [ ] Include all Child Packages

Object	Type	Target File
Account	table	
LineItem	table	
Order	table	
OrderStatus	table	
ShoppingBasket	table	
StockItem	table	
Transaction	table	

Note:

Alternatively you can select the **Project | Database Engineering | Generate Package DDL** menu option.

3. Select the checkbox against each inclusion required. Deselect the checkboxes for inclusions you do not require.

Notes:

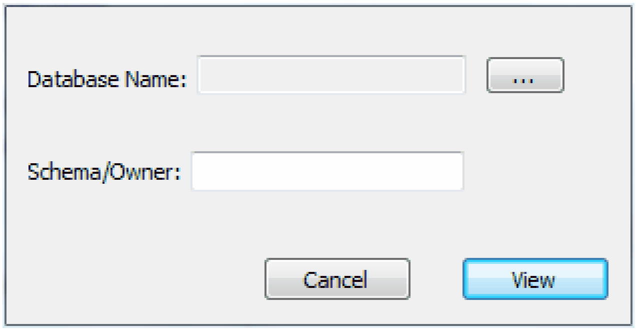
- Some checkboxes display only if the appropriate database is defined for the tables in the package. For example, **IF EXISTS** displays only if the database for the tables is PostgreSQL.
- If generating Oracle sequences, you must always select the **Generate Triggers** and **Generate Sequences** checkboxes; this ensures that a pre-insert trigger is generated to select the next sequence value to populate the column. Also select the **Auto Numbering** 130 checkbox 130 in the column properties.

4. To recursively generate DDL, select the **Include All Child Packages** checkbox.
5. Click on the **Generate** button to proceed. Enterprise Architect prompts you for file names as the process executes.

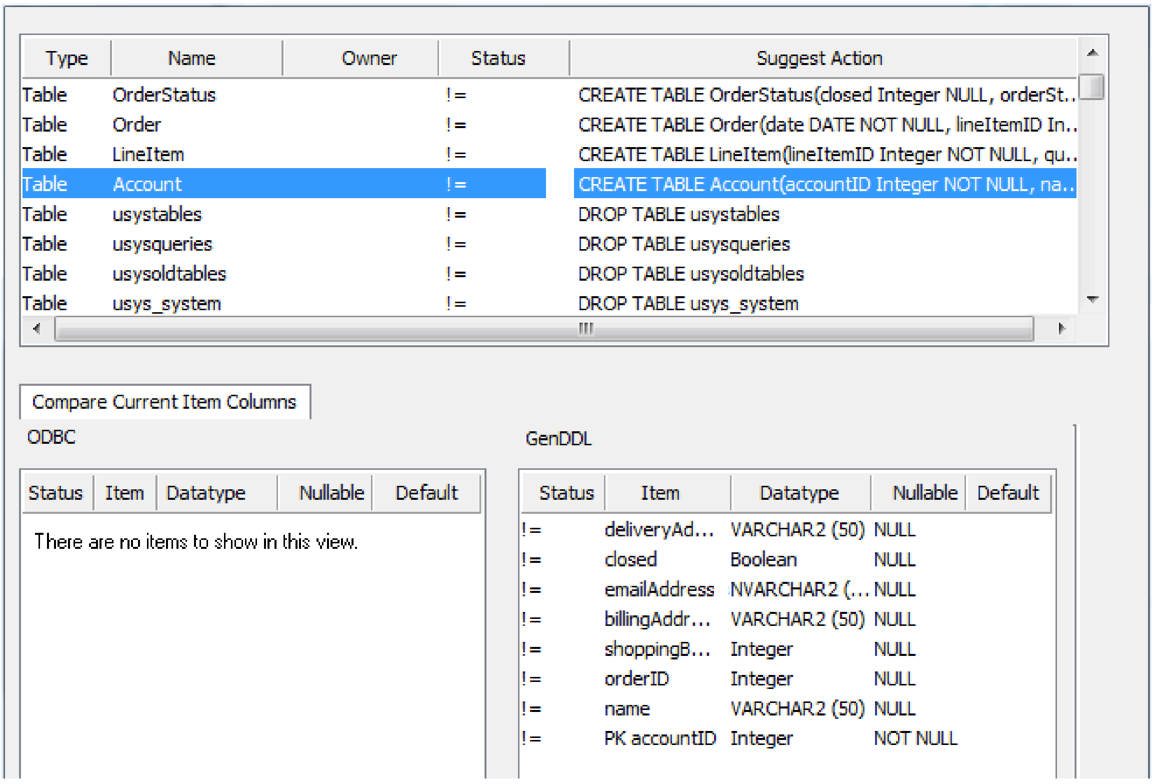
Compare DDL For a Database

When you have generated the DDL, you can compare it with the database. To do this, follow the steps below:

1. On the Generate Package DDL dialog, click on the **Compare** button. The Compare With Database dialog displays.



2. Click on the **[...]** button and locate the required database on the Select Data Source dialog.
3. For an Oracle database, if required you can also specify the Owner in the **Schema/Owner** field.
4. Click on the **View** button to perform the comparison. The Comparison Database dialog displays with the results of the comparison. Click on each table name to review information on that table.



Type	Name	Owner	Status	Suggest Action
Table	OrderStatus		!=	CREATE TABLE OrderStatus(closed Integer NULL, orderSt..
Table	Order		!=	CREATE TABLE Order(date DATE NOT NULL, lineItemID In..
Table	LineItem		!=	CREATE TABLE LineItem(lineItemID Integer NOT NULL, qu..
Table	Account		!=	CREATE TABLE Account(accountID Integer NOT NULL, na..
Table	usystables		!=	DROP TABLE usystables
Table	usysqueries		!=	DROP TABLE usysqueries
Table	usysoldtables		!=	DROP TABLE usysoldtables
Table	usys_system		!=	DROP TABLE usys_system

Compare Current Item Columns

ODBC

Status	Item	Datatype	Nullable	Default

There are no items to show in this view.

GenDDL

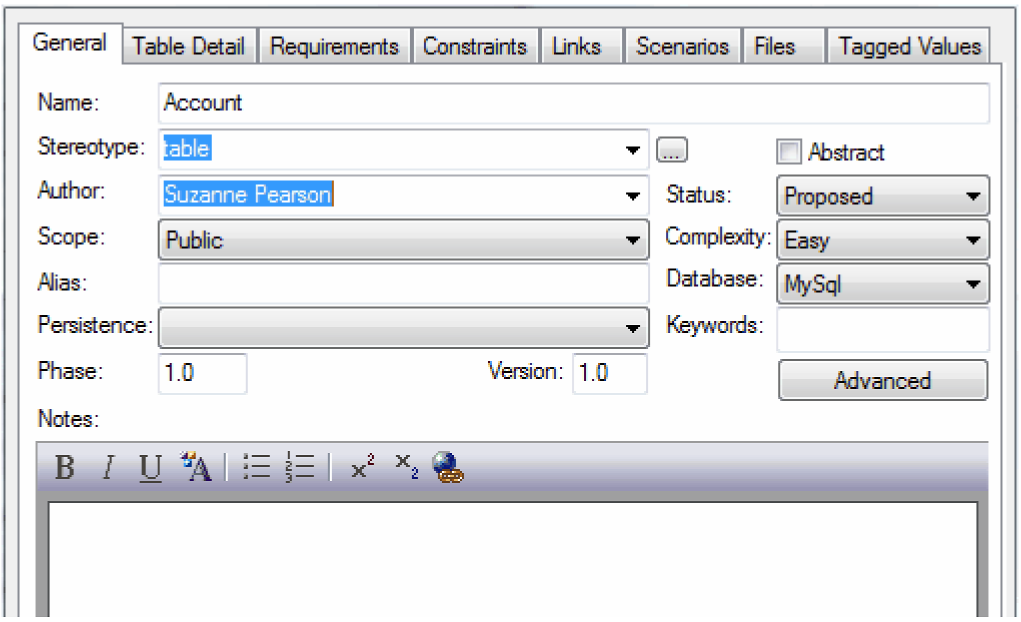
Status	Item	Datatype	Nullable	Default
!=	deliveryAd...	VARCHAR2 (50)	NULL	
!=	closed	Boolean	NULL	
!=	emailAddress	NVARCHAR2 (...	NULL	
!=	billingAddr...	VARCHAR2 (50)	NULL	
!=	shoppingB...	Integer	NULL	
!=	orderID	Integer	NULL	
!=	name	VARCHAR2 (50)	NULL	
!=	PK accountID	Integer	NOT NULL	

3.13 Data Type Conversion Procedure

Once a database schema has been set up on an Enterprise Architect diagram (either by importing through ODBC or manually setting up the tables), the DBMS can be changed to another type and the column datatypes are mapped accordingly.

To map the DBMS type of a table to another DBMS type, follow the steps below:

1. Double-click on the table element in a diagram to open the table Properties dialog.
2. The **Database** field shows the current DBMS for this table.
3. To map the column datatypes to another DBMS, select the target from the **Database** drop-down and click on the **Apply** button.
4. The datatypes are converted to match those of the new DBMS, and these are reflected in any DDL generated from this table.



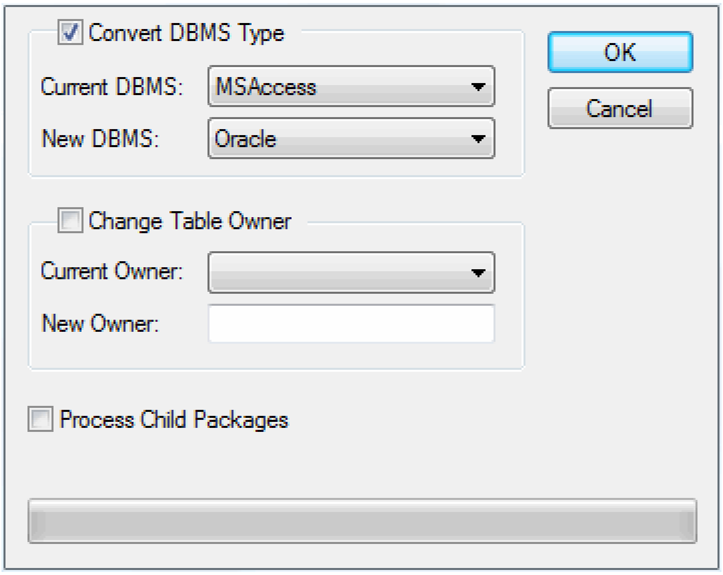
Name: Account
Stereotype: table
Abstract
Author: Suzanne Pearson
Status: Proposed
Scope: Public
Complexity: Easy
Alias:
Database: MySql
Persistence:
Keywords:
Phase: 1.0
Version: 1.0
Notes:

3.14 Data Type Conversion for a Package

The DBMS Package procedure or mapper enables you to convert a package of database tables from one DBMS type to another DBMS type, as well as providing the ability to change the ownership of tables.

To map the DBMS types of a package to another DBMS type, follow the steps below:

1. Right-click on the package in the Project Browser to display the context menu.
2. Select the **Code Engineering | Reset DBMS Options** menu option. The Manage DBMS Options dialog displays.



[✓] Convert DBMS Type

Current DBMS: MSAccess

New DBMS: Oracle

[ ] Change Table Owner

Current Owner:

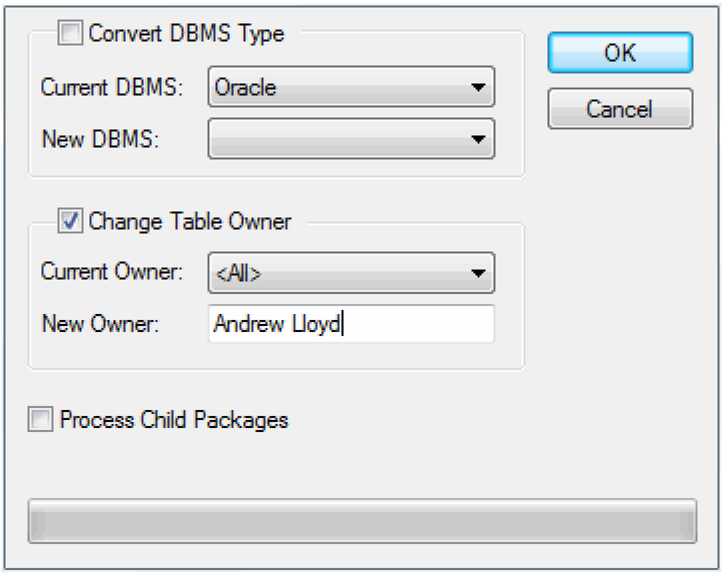
New Owner:

[ ] Process Child Packages

3. In the **Current DBMS** field, click on the drop-down arrow and select the current DBMS. In the **New DBMS** field click on the drop-down arrow and select the target DBMS.
4. Select the **Convert DBMS Type** checkbox.
5. If there are child packages that also require changing, select the **Process Child Packages** checkbox.
6. Click on the **OK** button. All tables in the selected packages are mapped to the new DBMS.

To change the owner of the table or all of the tables in a package, follow the steps below:

1. Right-click on the package in the Project Browser to display the context menu.
2. Select the **Code Engineering | Reset DBMS Options** menu option. The Manage DBMS Options dialog displays.



☐ Convert DBMS Type

Current DBMS: Oracle

New DBMS:

☑ Change Table Owner

Current Owner: <All>

New Owner: Andrew Lloyd

☐ Process Child Packages

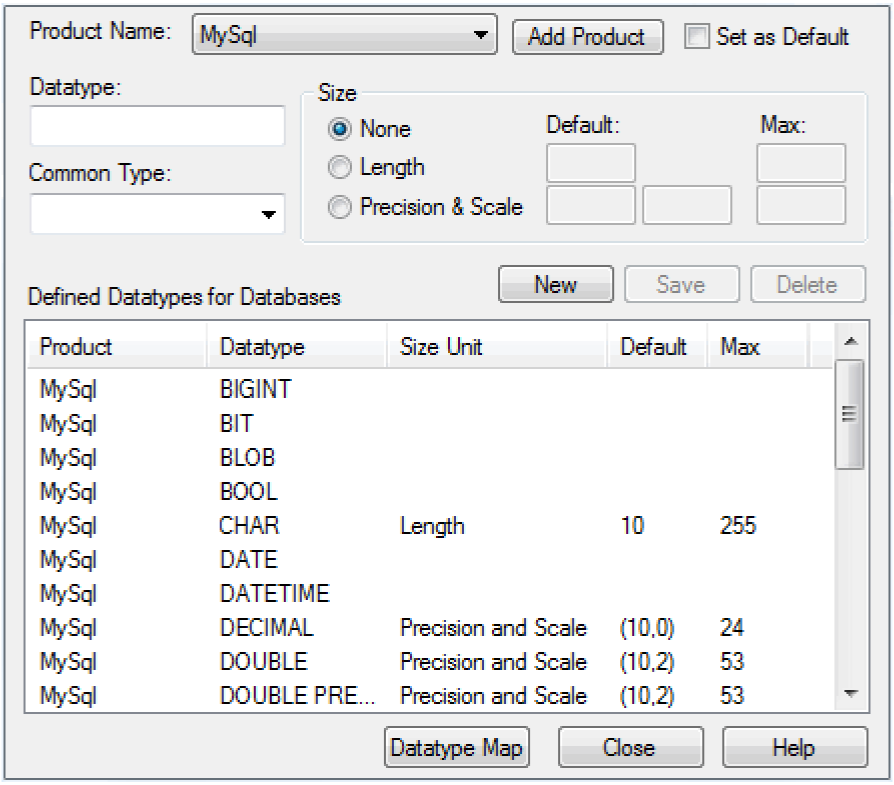
3. In the **New Owner** field, type the name for the new table owner.
4. In the **Current Owner** field, click on the drop-down arrow and select the current owner to change, or select **<All>** to change the ownership of all tables in the package to the name you typed in the **New Owner** field.
5. Select the **Change Table Owner** checkbox.
6. If there are child packages that also require changing, select the **Process Child Packages** checkbox.
7. Click on the **OK** button. The ownership changes for all Tables in the selected packages with the specified current owner.

For more information on setting the table owner see the *Set Table Owner* 126 topic. To display the table owner in the current diagram see the *Diagram Properties* topic in *UML Modeling with Enterprise Architect – UML Modeling Tool.*

3.15 DBMS Datatypes

When setting up your data modeling profile, you can customize the datatypes associated with a particular DBMS using the Database Datatypes screen. This screen enables you to add and configure custom data types. For some data types you must add the size and precision, defaults and maximum values.

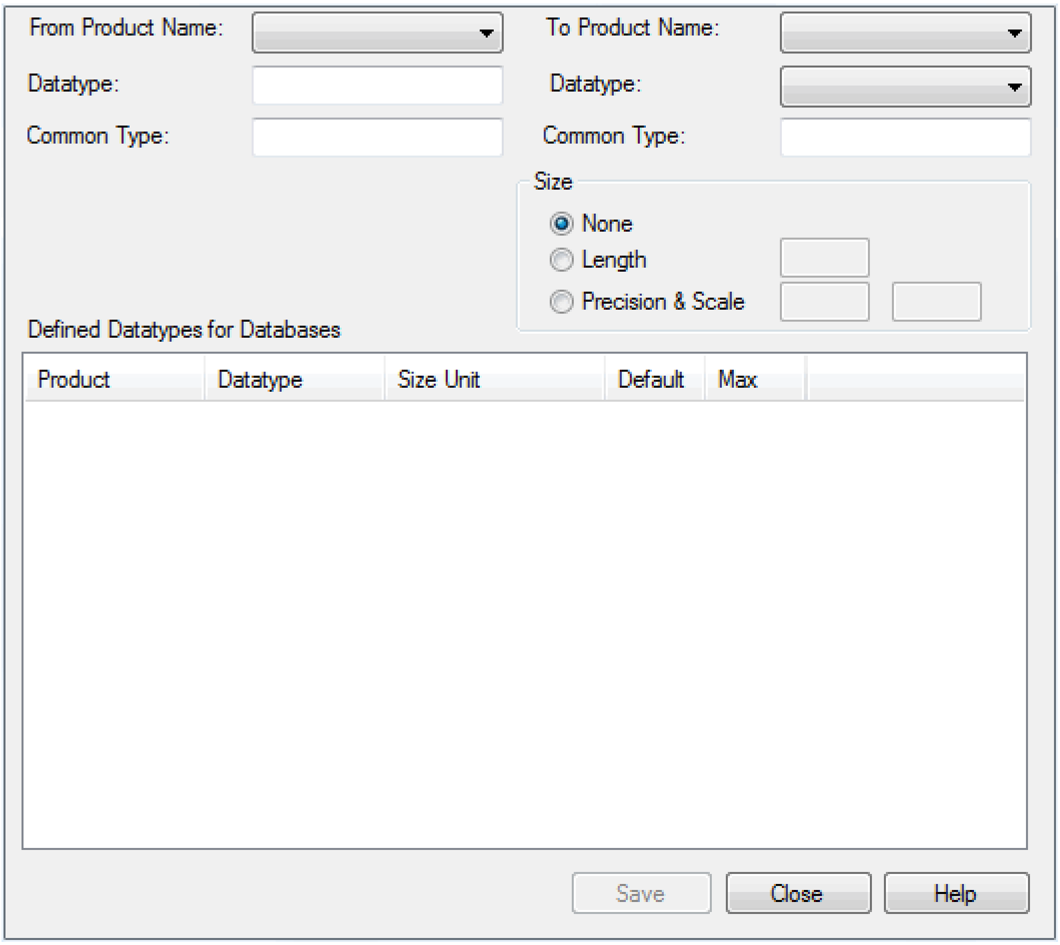
To access the Database Datatypes screen, select the **Settings | Database Datatypes** menu option. You can also add a DBMS product and configure the inbuilt data types.



Product	Datatype	Size Unit	Default	Max
MySql	BIGINT			
MySql	BIT			
MySql	BLOB			
MySql	BOOL			
MySql	CHAR	Length	10	255
MySql	DATE			
MySql	DATETIME			
MySql	DECIMAL	Precision and Scale	(10,0)	24
MySql	DOUBLE	Precision and Scale	(10,2)	53
MySql	DOUBLE PRE...	Precision and Scale	(10,2)	53

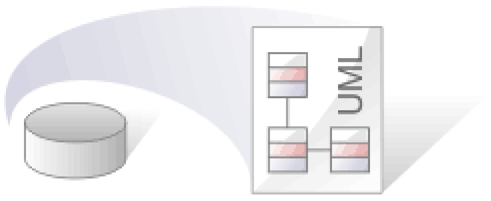
You can also map database datatype sizes between products. To do this, follow the steps below:

1. On the Database Datatypes dialog, click on the **Datatype Map** button. The Database Datatypes Mapping dialog displays.



2. In the **From Product Name** field, click on the drop-down arrow and select the DBMS product to map datatypes *from*. The Defined Datatypes for Databases panel displays all the defined datatypes for the product and, where appropriate, their sizes and values.
3. Click on the datatype to map (this must have a defined size unit and value). The **Datatype** and **Common type** fields under the **From Product Name** field display this datatype.
4. In the **To Product Name** field, click on the drop-down arrow and select the DBMS product to map datatypes *to*. The **Datatype** and **Common Type** fields under this field display the corresponding values to those in the fields for the *from* product.
5. In the Size panel, click on the radio button for the appropriate size unit and type the default values in the corresponding data fields.
6. Click on the **Save** button to save the mapping.
7. To map further datatypes, repeat this process from step 3.
8. When you have finished mapping datatypes, click on the **Close** button, and again on the Database Datatypes dialog.

3.16 Import Database Schema from ODBC



Analysis of legacy database systems is possible using Enterprise Architect's reverse engineering 4 capabilities. By connecting to a live database via ODBC, you can import the database schema into a standard UML model. Subsequent imports enable you to maintain synchronization between the data model and the live database.

Enterprise Architect supports importing database tables from an ODBC data source. Tables are imported as stereotyped Classes with suitable data definitions for the source DBMS.

Notes:

- Import of stored procedures and views is supported for: DB2; SQL Server; Firebird/Interbase; Informix; Ingres; Oracle 9i, 10g and 11g; MySQL; PostgreSQL; Sybase Adaptive Server Enterprise (ASE) and Sybase Adaptive Server Anywhere (ASA).
- If you are importing database schema from an MS Access Jet 4.0 database, please ensure that you have selected the **Use Jet 4.0** checkbox on the General page of the Options dialog (see *Using Enterprise Architect - UML Modeling Tool*). Otherwise, the Jet 3.5 routines are loaded. You must restart Enterprise Architect after selecting the checkbox.
- The ODBC connection should use the ODBC driver available from the DBMS vendor. For example, MySQL's ODBC driver for MySQL, and Oracle's ODBC driver for Oracle. Drivers provided by third-party vendors are not supported - this includes the *Microsoft* ODBC driver for Oracle.
- If setting up a ODBC connection for reverse engineering, the default settings are sufficient.
- Additional data types are available from the **Datamodeling Data Types** section of the Resources page on the Sparx Systems website.

Import Database Tables and Stored Procedures

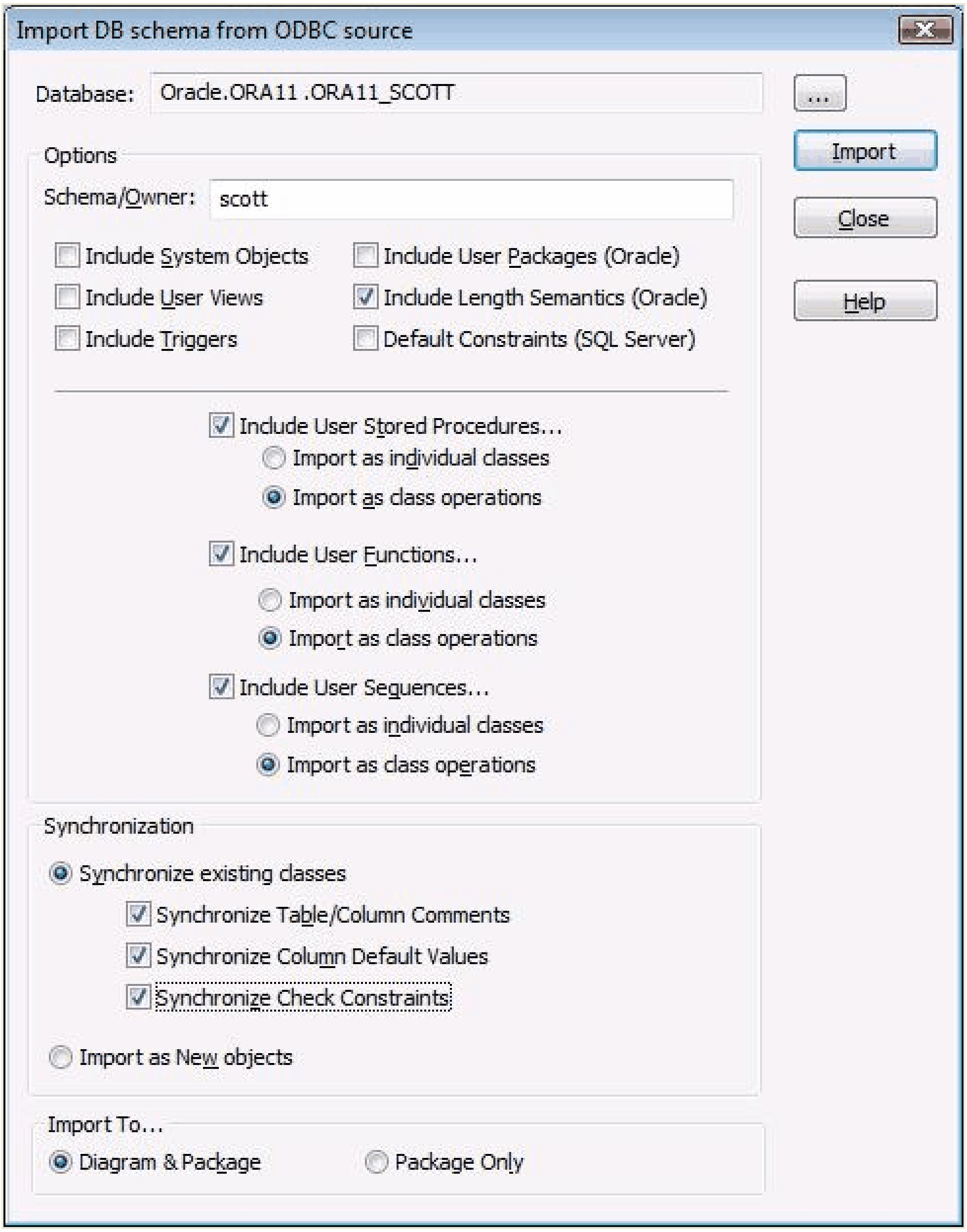
To import database tables and stored procedures, follow the steps below:

1. Select any package in the Logical View.
2. To import into:
   - The package only, right-click on the package to display the context menu, and select the **Code Engineering | Import DB Schema from ODBC** menu option.
   - A diagram, right-click on the diagram in the selected package to open the context menu, and select the **Import DB schema from ODBC** menu option.

   Note:

   Alternatively you can select the **Project | Database Engineering | Import DB Schema from ODBC** menu option.

   The Import DB Schema from ODBC Source dialog displays.


Import DB schema from ODBC source

Database: Oracle.ORA11 .ORA11_SCOTT

Options

Schema/Owner: scott

Include System Objects
Include User Views
Include Triggers
Include User Packages (Oracle)
[✓ Include Length Semantics (Oracle)]
Default Constraints (SQL Server)

[✓ Include User Stored Procedures...]
Import as individual classes
[✓ Import as class operations]

[✓ Include User Functions...]
Import as individual classes
[✓ Import as class operations]

[✓ Include User Sequences...]
Import as individual classes
[✓ Import as class operations]

Synchronization

[✓ Synchronize existing classes]
[✓ Synchronize Table/Column Comments]
[✓ Synchronize Column Default Values]
[✓ Synchronize Check Constraints]

Import as New objects

Import To...

[✓ Diagram & Package] Package Only

3. In the **Database** field, click on the **[...]** (Browse) button and select a suitable ODBC data source 162 from the ODBC dialog (ODBC must be installed and configured on your machine for this to work correctly).

   When you have selected the data source, the **Database** field shows the DBMS, the database server ID and the database name, separated by full stops; that is: *dbms.dbserver.database*.
4. If importing from Oracle, to restrict the import to a specific owner, type the owner name in the **Schema/Owner** field. By default, Enterprise Architect inserts the Oracle user name in this field.

   For imports from other types of database, leave this field blank.
5. In the Filter panel, select the appropriate checkboxes for additional items to include in the import.

   Select the appropriate checkboxes to import system tables and views, user views, triggers and/or Oracle packages.

   If you select to import *User Functions* and/or *User Sequences* as individual Classes, then they are imported as separate elements and the Properties dialog is solely concerned with the Function or Sequence definition. For *Stored Procedures*, always select this option

If you select to import *User Functions* and/or *User Sequences* as Class operations, then they are imported as operations (methods) and you view and edit them through the Operations Properties dialog of the parent Class.

6. When synchronizing existing Classes, select the appropriate checkbox in the Synchronization panel to determine whether the model comments, default values or constraints are to be synchronized with the ODBC tables, or as new objects.

   Note:

   It is only possible to import into a diagram if it is in the selected package. If a diagram from another package is open, a message displays to give the option to cancel the import or to continue importing into the package only. The Import DB Schema from ODBC Source dialog includes checkbox options to import into the diagram and package, or into the package only.

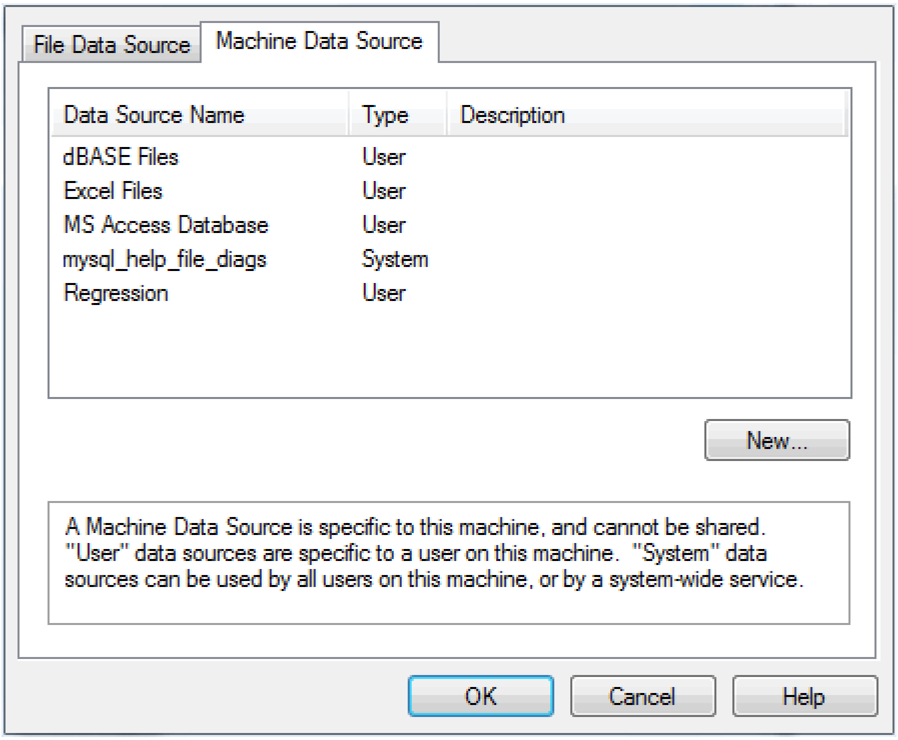
   If no diagram is open, the **Package Only** radio button defaults to selected and the options are disabled. If the open diagram is in the selected package, you can select either option.

7. Click on the **Import** button to start the import.
8. Select the tables 163 and - if appropriate - stored procedures to import.

This completes the procedure. See the *Imported Class Elements* 163 topic.

3.16.1 Select a Data Source

To import DDL from existing data sources, you must have a suitable ODBC connection installed and configured (see *UML Model Management*). From the Import DB Schema from ODBC Source dialog you can select the ODBC data source using the standard windows ODBC set-up dialog. Click on the data source name and then click on the **OK** button.



Data Source Name	Type	Description
dBASE Files	User	
Excel Files	User	
MS Access Database	User	
mysql_help_file_diags	System	
Regression	User	

A Machine Data Source is specific to this machine, and cannot be shared. "User" data sources are specific to a user on this machine. "System" data sources can be used by all users on this machine, or by a system-wide service.

3.16.2 Select Tables

When you have opened the ODBC data source, Enterprise Architect acquires a list of tables and stored procedures suitable for importing. This is presented in a list form for you to select from.

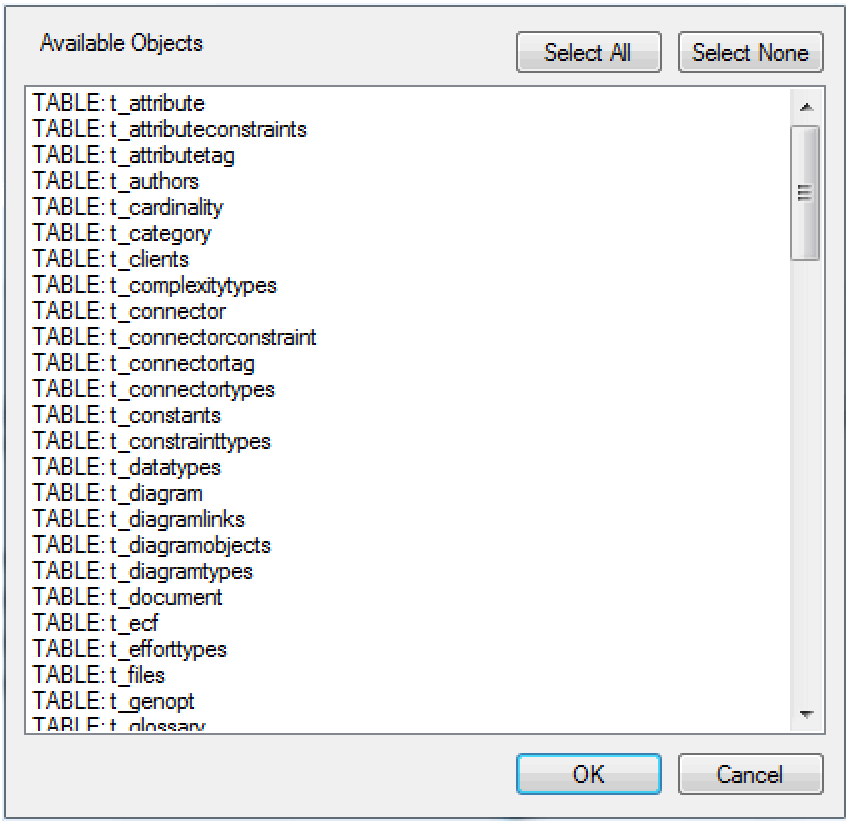


TABLE: t_attribute
TABLE: t_attributeconstraints
TABLE: t_attributetag
TABLE: t_authors
TABLE: t_cardinality
TABLE: t_category
TABLE: t_clients
TABLE: t_complexitytypes
TABLE: t_connector
TABLE: t_connectorconstraint
TABLE: t_connectortag
TABLE: t_connectortypes
TABLE: t_constants
TABLE: t_constrainttypes
TABLE: t_datatypes
TABLE: t_diagram
TABLE: t_diagramlinks
TABLE: t_diagramobjects
TABLE: t_diagramtypes
TABLE: t_document
TABLE: t_ecf
TABLE: t_efforttypes
TABLE: t_files
TABLE: t_genopt
TABLE: t_glossary

Highlight the tables and stored procedures to import and clear those you do not require.

Selection shortcuts:

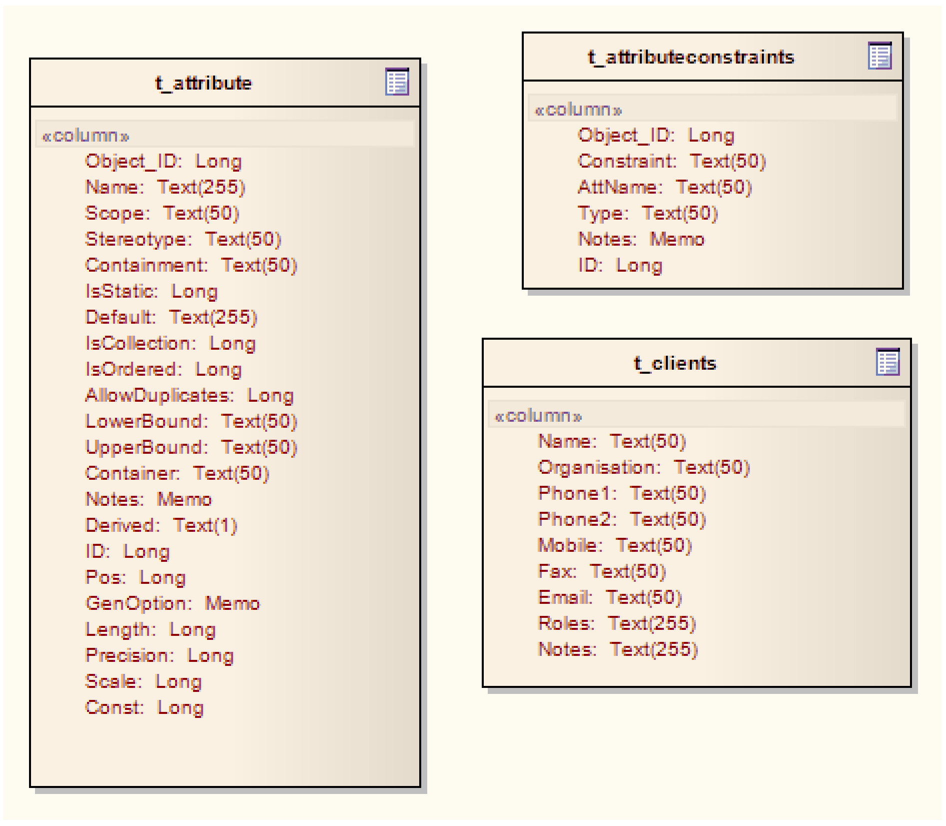
- To select all tables and procedures, click on the **Select All** button
- To clear all tables and procedures, click on the **Select None** button
- Hold down **[Ctrl]** while clicking on tables and procedures to select multiple objects
- Hold down **[Shift]** and click on tables and procedures to select a range.

When you have selected the tables and procedures, click on the **OK** button.

3.16.3 The Imported Class Elements

When you import DDL table definitions they are converted to stereotyped Classes according the *UML Data Modeling Profile*.

The image below shows some example tables imported into the model using an ODBC data connection.



t_attribute

«column»
Object_ID: Long
Name: Text(255)
Scope: Text(50)
Stereotype: Text(50)
Containment: Text(50)
IsStatic: Long
Default: Text(255)
IsCollection: Long
IsOrdered: Long
AllowDuplicates: Long
LowerBound: Text(50)
UpperBound: Text(50)
Container: Text(50)
Notes: Memo
Derived: Text(1)
ID: Long
Pos: Long
GenOption: Memo
Length: Long
Precision: Long
Scale: Long
Const: Long

t_attributeconstraints

«column»
Object_ID: Long
Constraint: Text(50)
AttName: Text(50)
Type: Text(50)
Notes: Memo
ID: Long

t_clients

«column»
Name: Text(50)
Organisation: Text(50)
Phone1: Text(50)
Phone2: Text(50)
Mobile: Text(50)
Fax: Text(50)
Email: Text(50)
Roles: Text(255)
Notes: Text(255)

Index

Code Engineering Using UML Models

www.sparxsystems.com