



## Software Developers' Kit

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

*This booklet describes the facilities of the Software Developers Kit, which enables you to customize and extend the facilities of Enterprise Architect.*



# Enterprise Architect Software Developers' Kit

© 1998-2009 Sparx Systems Pty Ltd

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: October 2009

## **Publisher**

*Sparx Systems*

## **Managing Editor**

*Geoffrey Sparks*

## **Technical Editors**

*Geoffrey Sparks*

*Salvatore Mancarella*

*Neil Capey*

## **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	1
<b>SDK for Enterprise Architect</b>	<b>2</b>
<b>Developing Profiles</b>	<b>3</b>
<b>Custom Stereotypes</b>	<b>3</b>
<b>Create Profiles</b>	<b>5</b>
Create a Profile Package	5
Add Stereotypes and Metaclasses	6
Define Stereotype Tagged Values	8
With Predefined Tag Types	9
With Supported Attributes	9
Use the Tagged Value Connector	10
Define Stereotype Constraints	11
Add Enumeration Elements	13
Add Shape Scripts	14
Set Default Appearance	16
Export a UML Profile	16
Save Profile Options	17
Supported Attributes	18
Define a Stereotype as a Metatype	19
Define Multiple-Stereotype Level	19
Define Creation of Instance	20
Create Composite Elements	21
Define Child Diagram Types	21
Stereotypes Profiles	22
<b>Quick Linker</b>	<b>23</b>
Quick Linker Definition Format	23
Quick Linker Example	25
Hide Default Quick Linker Settings	26
Quick Linker Object Names	27
<b>MDG Technologies in SDK</b>	<b>28</b>
<b>Create MDG Technologies</b>	<b>28</b>
Add a Profile	33
Add a Pattern	33
Add a Diagram Profile	34
Add a Toolbox Profile	35
Add Task Panel Pages	36
Add Tagged Value Types	37
Add Code Modules	38
Add MDA Transforms	40
Add Images	41
Add Scripts	41
Add RTF Report Templates	42
Add Linked Document Templates	43
<b>Working with MTS Files</b>	<b>44</b>
<b>Customize Toolbox Profiles</b>	<b>45</b>
Create Toolbox Profiles	45
Toolbox Page Attributes	46
Create Hidden Sub-Menus	46
Override Default Toolboxes	47
Assign Icons To Toolbox Items	47
Enterprise Architect Toolboxes	48
Elements Used in Toolboxes	48

Connectors Used In Toolboxes .....	49
<b>Create Diagram Profiles .....</b>	<b>50</b>
Built-In Diagram Types.....	51
Attribute Values - stylex & pdata .....	51
<b>Create Tasks Pane Profiles .....</b>	<b>52</b>
Define Tasks Pane Toolboxes.....	53
Built-In Tasks Pane Commands.....	53
Run Add-In Functions.....	54
Define Tasks Pane Contexts.....	55
Allocate Tasks Pane Contexts.....	55
Save a Tasks Pane Profile.....	56
<b>Define Validation Configuration .....</b>	<b>56</b>
<b>Incorporate Model Templates .....</b>	<b>57</b>
<b>Deploy An MDG Technology .....</b>	<b>57</b>
<b>Shape Scripts .....</b>	<b>59</b>
<b>Getting Started With Shape Scripts .....</b>	<b>59</b>
<b>Shape Editor .....</b>	<b>62</b>
<b>Write Scripts .....</b>	<b>63</b>
Syntax Grammar .....	63
Shape Attributes.....	64
Drawing Methods.....	66
Color Queries.....	70
Conditional Branching.....	70
Query Methods.....	70
Display Item Properties.....	70
Sub-Shapes.....	73
Reserved Names.....	74
Miscellaneous.....	74
<b>Example Scripts .....</b>	<b>75</b>
<b>Tagged Value Types .....</b>	<b>79</b>
<b>Predefined Structured Types .....</b>	<b>79</b>
<b>Create Structured Tagged Values .....</b>	<b>81</b>
<b>Predefined Reference Data Types .....</b>	<b>82</b>
<b>Create Reference Data Tagged Values .....</b>	<b>83</b>
<b>Create Custom Tagged Value Type .....</b>	<b>84</b>
<b>Code Template Framework in SDK .....</b>	<b>86</b>
<b>Code Template Syntax .....</b>	<b>86</b>
Literal Text.....	86
Macros.....	87
Template Substitution Macros .....	87
Field Substitution Macros.....	88
Tagged Value Macros.....	100
Function Macros.....	101
Control Macros.....	104
EASL Code Generation Macros.....	107
EASL Collections.....	109
EASL Properties.....	111
Variables.....	115
<b>The Code Template Editor in SDK .....</b>	<b>116</b>
Custom Templates.....	117
Override Default Templates.....	118
Add New Stereotyped Templates.....	119
Create Custom Language Template .....	120
<b>Enterprise Architect Add-In Model .....</b>	<b>121</b>
<b>Add-In Tasks .....</b>	<b>122</b>
Create Add-Ins.....	122
Define Menu Items.....	122

Deploy Add-Ins.....	123
Tricks and Traps.....	124
<b>The Add-In Manager .....</b>	<b>126</b>
<b>Add-In Search .....</b>	<b>126</b>
XML Format (Search Data).....	127
<b>Add-In Events .....</b>	<b>127</b>
EA_Connect.....	128
EA_Disconnect.....	128
EA_GetMenuItems.....	129
EA_GetMenuState.....	129
EA_MenuClick.....	130
EA_OnOutputItemClicked.....	131
EA_OnOutputItemDoubleClicked.....	131
EA_ShowHelp.....	132
<b>Broadcast Events .....</b>	<b>133</b>
EA_FileOpen.....	133
EA_FileClose.....	134
EA_FileNew.....	134
Pre-Deletion Events.....	135
EA_OnPreDeleteElement.....	135
EA_OnPreDeleteConnector.....	135
EA_OnPreDeleteDiagram.....	136
EA_OnPreDeletePackage.....	136
Pre-New Events.....	137
EA_OnPreNewElement.....	137
EA_OnPreNewConnector.....	138
EA_OnPreNewDiagramObject.....	139
EA_OnPreNewAttribute.....	139
EA_OnPreNewMethod.....	140
EA_OnPreNewPackage.....	141
EA_OnPreExitInstance.....	141
Post-New Events.....	142
EA_OnPostNewElement.....	142
EA_OnPostNewConnector.....	142
EA_OnPostNewDiagramObject.....	143
EA_OnPostNewAttribute.....	144
EA_OnPostNewMethod.....	144
EA_OnPostNewPackage.....	145
EA_OnPostInitialized.....	145
EA_OnPostTransform.....	146
Technology Events.....	146
EA_OnInitialize_Technologies.....	147
EA_OnPreActivateTechnology.....	147
EA_OnPostActivateTechnology.....	148
EA_OnPreDeleteTechnology.....	148
EA_OnDeleteTechnology.....	149
EA_OnImportTechnology.....	149
Context Item Events.....	150
EA_OnContextItemChanged.....	150
EA_OnContextItemDoubleClicked.....	151
EA_OnNotifyContextItemModified.....	152
Compartment Events.....	153
EA_QueryAvailableCompartments.....	153
EA_GetCompartmentData.....	153
Model Validation Broadcasts.....	155
EA_OnInitializeUserRules.....	155
EA_OnStartValidation.....	156
EA_OnEndValidation.....	156

EA_OnRunElementRule.....	156
EA_OnRunPackageRule.....	157
EA_OnRunDiagramRule.....	157
EA_OnRunConnectorRule.....	158
EA_OnRunAttributeRule.....	158
EA_OnRunMethodRule.....	159
EA_OnRunParameterRule.....	159
Model Validation Example.....	160
EA_OnRetrieveModelTemplate.....	163
<b>Custom Views .....</b>	<b>164</b>
Create a Custom View.....	164
<b>MDG Add-Ins .....</b>	<b>165</b>
MDG Events.....	165
MDGBuild Project.....	166
MDGConnect.....	166
MDGDisconnect.....	167
MDGGetConnectedPackages.....	167
MDGGetProperty.....	168
MDGMerge.....	169
MDGNewClass.....	170
MDGPostGenerate.....	171
MDGPostMerge.....	171
MDGPreGenerate.....	172
MDGPreMerge.....	172
MDGPreReverse.....	173
MDGRunExe.....	174
MDGView.....	174
<b>Enterprise Architect Object Model .....</b>	<b>176</b>
<b>Using the Automation Interface .....</b>	<b>176</b>
Connect to the Interface.....	176
Set References In Visual Basic.....	178
Examples and Tips.....	179
Call from Enterprise Architect.....	180
Available Resources.....	181
<b>Reference .....</b>	<b>181</b>
Interface Overview.....	182
App.....	185
Enumerations.....	186
ConstLayoutStyles Enum.....	186
CreateModelType Enum.....	186
EAEditionTypes Enum.....	187
EnumRelationSetType Enum.....	187
MDGMenus Enum.....	187
ObjectType Enum.....	188
PropType Enum.....	188
ReloadType Enum.....	189
XMISet Enum.....	189
Repository.....	189
Repository.....	190
Author .....	203
Client .....	204
Collection .....	205
Datatype .....	206
EventProperties.....	207
EventProperty.....	208
ModelWatcher.....	208
Package .....	209
ProjectIssues.....	212

ProjectResource.....	213
PropertyType.....	214
Reference.....	215
Stereotype.....	216
Task .....	216
Term .....	217
Element.....	218
Constraint.....	220
Effort .....	220
Element .....	221
File .....	228
Issue (Maintenance).....	228
Metric .....	229
Requirement.....	230
Resource .....	231
Risk .....	232
Scenario .....	232
TaggedValue.....	233
Test .....	234
Element Features.....	235
Attribute .....	235
AttributeConstraint.....	237
AttributeTag.....	238
CustomProperties.....	239
EmbeddedElements.....	239
Method .....	240
MethodConstraint.....	242
MethodTag.....	243
Parameter.....	243
Partitions .....	244
Properties.....	245
Transitions.....	246
Connector.....	246
ConnectorConstraint.....	247
Connector.....	248
ConnectorEnd.....	251
ConnectorTag.....	252
RoleTag .....	253
Diagram.....	254
Diagram .....	255
DiagramLinks.....	258
DiagramObjects.....	259
SwimlaneDef.....	260
Swimlanes.....	261
Swimlane .....	262
Project Interface.....	262
Project .....	262
Code Samples.....	273
Open the Repository.....	273
Iterate Through a .EAP File.....	274
Add and Manage Packages.....	274
Add and Manage Elements.....	275
Add a Connector.....	275
Add and Manage Diagrams.....	276
Add and Delete Features.....	277
Element Extras.....	277
Repository Extras.....	280
Stereotypes.....	281

Work with Attributes.....	282
Work with Methods.....	282

**Index**

**284**



---

# Foreword

This user guide describes the facilities of the Software Developers Kit, which enables you to customize and extend the facilities of Enterprise Architect.

## SDK for Enterprise Architect



### Introduction

Welcome to the *Enterprise Architect Software Developers Kit (SDK)*. This is a special section of the *Enterprise Architect User Guide*, covering the more advanced aspects of extending and customizing Enterprise Architect.

In describing aspects of extending Enterprise Architect, it is expected that you are familiar with the concepts introduced in the main body of the *Enterprise Architect User Guide*. Wherever appropriate, cross-references to these concepts are provided in the text.

### Contents

- [UML Profiles](#) <sup>[3]</sup> (incorporating [UML Stereotypes](#) <sup>[3]</sup>)
- [MDG Technologies](#) <sup>[28]</sup>
- [Shape Scripts](#) <sup>[59]</sup>
- [Tagged Value Types](#) <sup>[79]</sup>
- [Code Template Framework](#) <sup>[86]</sup>
- [Enterprise Architect Object Model \(Automation Interface\)](#) <sup>[176]</sup>
- [Enterprise Architect Add-In Model](#) <sup>[121]</sup>

## 1 Developing Profiles



### Introduction

UML Profiles provide a means of extending the UML Language, which enables you to build UML models in particular domains. They are based on additional [stereotypes](#)<sup>[3]</sup> and [Tagged Values](#)<sup>[79]</sup> that are applied to UML elements, connectors and their components. A Profile is a collection of such extensions that together describe some particular modeling problem and facilitate modeling constructs in that domain. UML Profiles for Enterprise Architect are specified in XML files, with a specific format. These XML files can be imported into Enterprise Architect through the [Resources](#) window.

The imported Profile also automatically generates a page of elements and relationships in the Enterprise Architect UML [Toolbox](#).

The [Resources](#) window contains a tree structure with entries for items such as MDG Technologies, Documents, Stylesheets, Matrix profiles and UML Profiles. The *UML Profiles* node initially contains no entries; to be able to use Profiles you must import them into Enterprise Architect from supplied XML files.

Items in the Profile represent stereotypes. UML supports a large number of stereotypes, which are an inbuilt mechanism for logically extending or altering the meaning, display, appearance and syntax of a model element. Different model elements have different stereotypes associated with them.

For more information on the use of Profiles in Enterprise Architect, see the *UML Profiles* topic in *Extending UML With Enterprise Architect*.

For information on developing your own Profiles, see the following topics:

- [Custom Stereotypes](#)<sup>[3]</sup>
- [Create Profiles](#)<sup>[5]</sup>
- [Quick Linker](#)<sup>[23]</sup>
- [Toolbox Profiles](#)<sup>[45]</sup>
- [Diagram Profiles](#)<sup>[50]</sup>
- [Task Pane Profiles](#)<sup>[52]</sup>

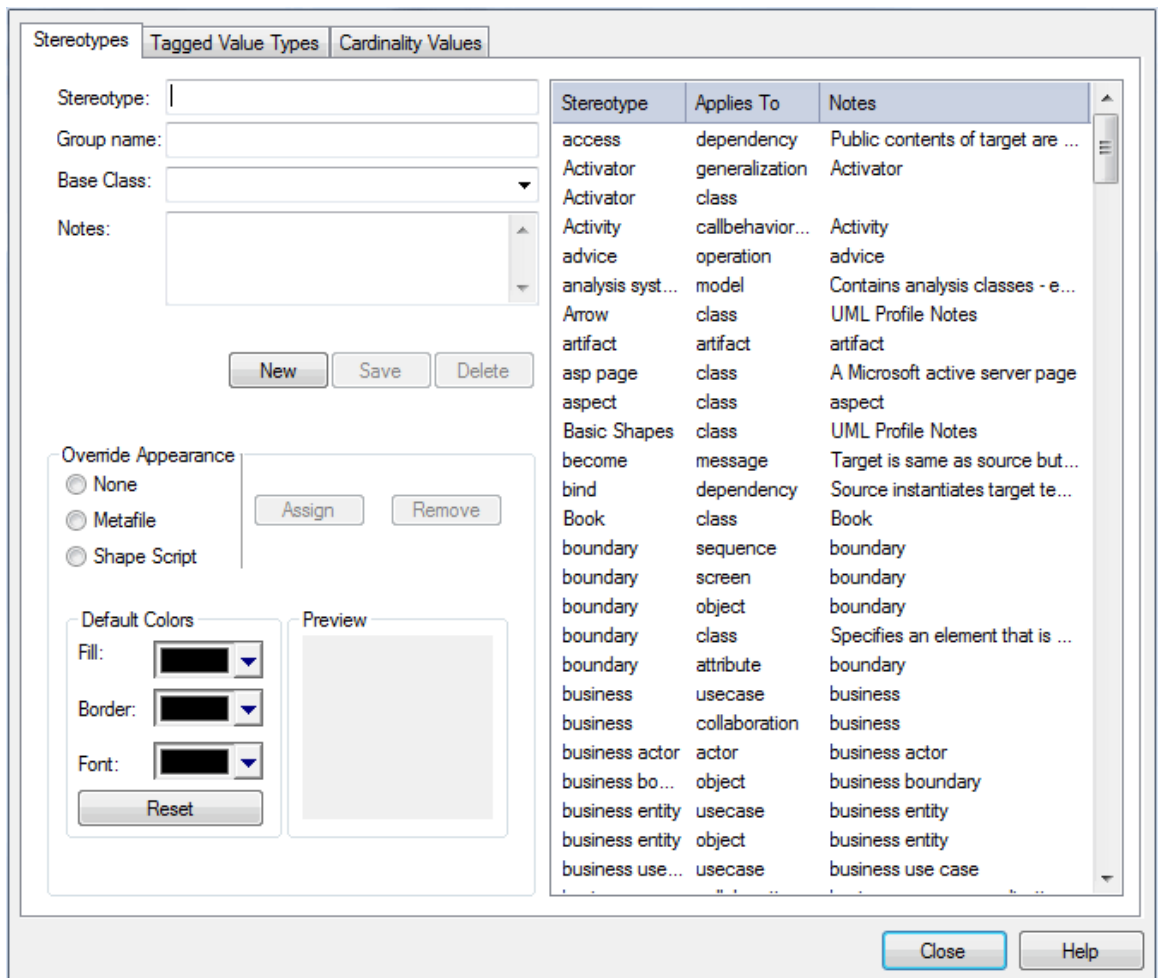
### 1.1 Custom Stereotypes

UML supports a large number of *stereotypes*, which are an inbuilt mechanism for logically extending or altering the meaning, display and syntax of a model element. Different model elements have different stereotypes associated with them. For more information on the use of stereotypes in Enterprise Architect, see the *UML Stereotypes* topic in *Extending UML With Enterprise Architect*.

In Enterprise Architect you can create new stereotypes with their own custom appearance. The stereotypes can be altered to make use of metafiles (image files) and customized colors, or you can make use of the Enterprise Architect Shape Script to make new element shapes to determine the shape and dimensions of the element.

To add your own custom stereotypes, follow the steps below:

1. From the main menu, select **Settings | UML**. The **UML Types** dialog displays, defaulted to the **Stereotypes** tab.



2. Type or select a **Stereotype** name.
3. Select a **Base Class** from the drop-down list.
4. To associate a Metafile with this stereotype, click on the **Metafile** radio button and the **Assign** button, and locate the required .emf or .wmf file.
5. Enter optional **Notes** and select **Default Colors** for this stereotype.
6. Click on the **Save** button to save the stereotype.

The table below describes the functionality of the **Stereotypes** tab.

Option	Use to
<b>Stereotype</b>	Specify the name of the stereotype.
<b>Group name</b>	Enable grouping of stereotype features by a plural name, for attributes and operations, which is shown on diagrams in the attribute and operations compartments.
<b>Base Class</b>	Enable the stereotyped element to inherit the base characteristics from a pre-existing element type.
<b>Notes</b>	Add any stereotype notes.
<b>Override Appearance</b>	
<b>None</b>	Switch to the default element appearance.
<b>Metafile</b>	Enable an image file to be used for the appearance of the stereotype.
<b>Shape Script</b>	Specify custom shapes for the stereotype using the <i>Enterprise Architect Shape Scripting</i> language. For more information see the <a href="#">Shape Scripting</a> <sup>[59]</sup> topic.

Option	Use to
<b>Assign</b>	Add the associated metafile or Shape Script from the stereotyped element.
<b>Remove</b>	Remove the associated metafile or Shape Script from the stereotyped element.
<b>Default Colors</b>	
<b>Fill</b>	Set the default background color of the element.
<b>Border</b>	Control the border color.
<b>Font</b>	Control the color of the stereotype font.
<b>Reset</b>	Reset the appearance of the element to the default element appearance.

**Note:**

You can transport these custom stereotype definitions 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*.

## 1.2 Create Profiles

This topic describes how to create profiles and profile items. These creation tasks include creating the profile stereotypes, defining the metaclasses they apply to, and defining Tagged Values and constraints. This topic also describes how to export a profile for use in UML modeling.

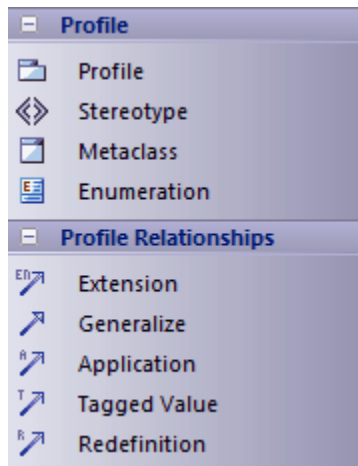
To create a UML Profile, follow the steps below:

1. [Create a Profile Package](#) <sup>5</sup>
2. [Add Stereotypes and Metaclasses](#) <sup>6</sup>
3. [Define Tagged Values for Stereotypes](#) <sup>8</sup>
4. [Define Constraints for Stereotypes](#) <sup>11</sup>
5. [Add Enumerations](#) <sup>13</sup>
6. [Add Shape Scripts](#) <sup>14</sup>
7. [Set Default Appearance](#) <sup>16</sup>
8. [Export the Profile](#) <sup>16</sup>

### 1.2.1 Create a Profile Package

In Enterprise Architect, you must create a UML Profile in a Package that has the stereotype «*profile*». To create a Profile Package, follow the steps below.

1. Open or create a Package diagram.
2. Open the **Profile** page of the Enterprise Architect UML **Toolbox (More tools | Profile)**.

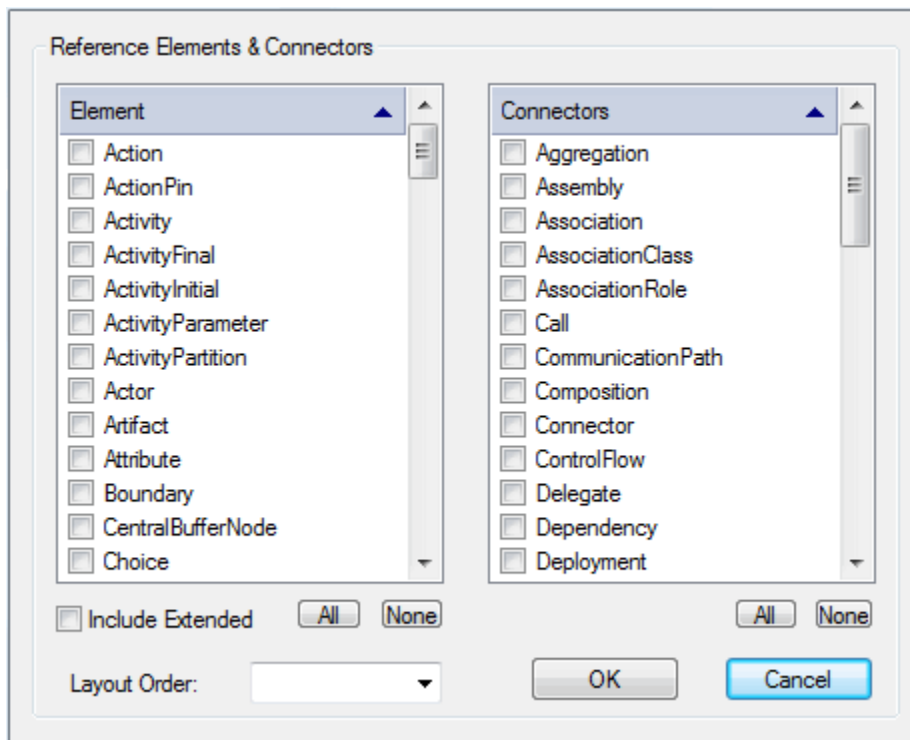


3. Drag the *Profile* item onto the Class diagram. The **New Model Package** dialog displays.
  4. In the **Package Name** field, type a name for the Profile.
  5. Select the **Automatically add new diagram** checkbox.
  6. Click on the **OK** button. The **New Diagram** dialog displays (see *UML Modeling With Enterprise Architect - UML Modeling Tool*).
  7. Provide the required diagram name, and select the diagram group **UML Structural** and diagram type **Class**.
  8. Click on the **OK** button. Enterprise Architect creates a package with the stereotype «*profile*» and with a child Class diagram.
  9. Double-click on the Profile Package on the diagram to open the child diagram.
- You now use this child diagram to [add stereotypes](#) <sup>F6</sup> to the Profile.

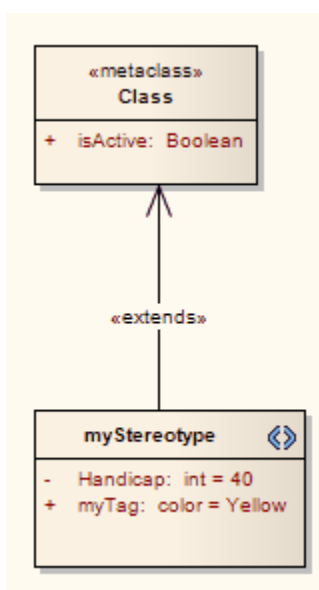
### 1.2.2 Add Stereotypes and Metaclasses

To add metaclasses and stereotypes to a Profile, follow the steps below for as many stereotypes and metaclasses as you require:

1. Open the child diagram of the Profile Package.
2. Drag the *Metaclass* element from the **Profile** page of the Enterprise Architect UML **Toolbox** onto the diagram. The **Create New Metaclass** dialog displays, in which you can tick multiple metaclasses for dropping onto the diagram.



3. Scroll down the **Element** list and select the checkbox for **Class**.
4. Click on the **OK** button, display the Class **Properties** dialog, and in the **Name** field type a name for the element. Click on the **OK** button again.
5. Drag a **Stereotype** element from the **Toolbox** onto the diagram. If the **Properties** dialog does not display, double-click on the element on the diagram.
6. In the **Name** field, type a name for the stereotype.
7. Click on the **OK** button and, if it displays, **Close** the **Generate Code** dialog.
8. Click on the **Extension** relationship in the **Toolbox** and drag the connection from the stereotype element to the metaclass element.
9. Your diagram should now resemble the one below:



You can now add [stereotype Tags](#)<sup>[8]</sup>, [Constraints](#)<sup>[11]</sup>, [Enumerations](#)<sup>[13]</sup>, and/or [Shape Scripts](#)<sup>[14]</sup> to your

Profile, and define the [default appearance](#)<sup>[16]</sup> of the elements or connectors as required.

### 1.2.3 Define Stereotype Tagged Values

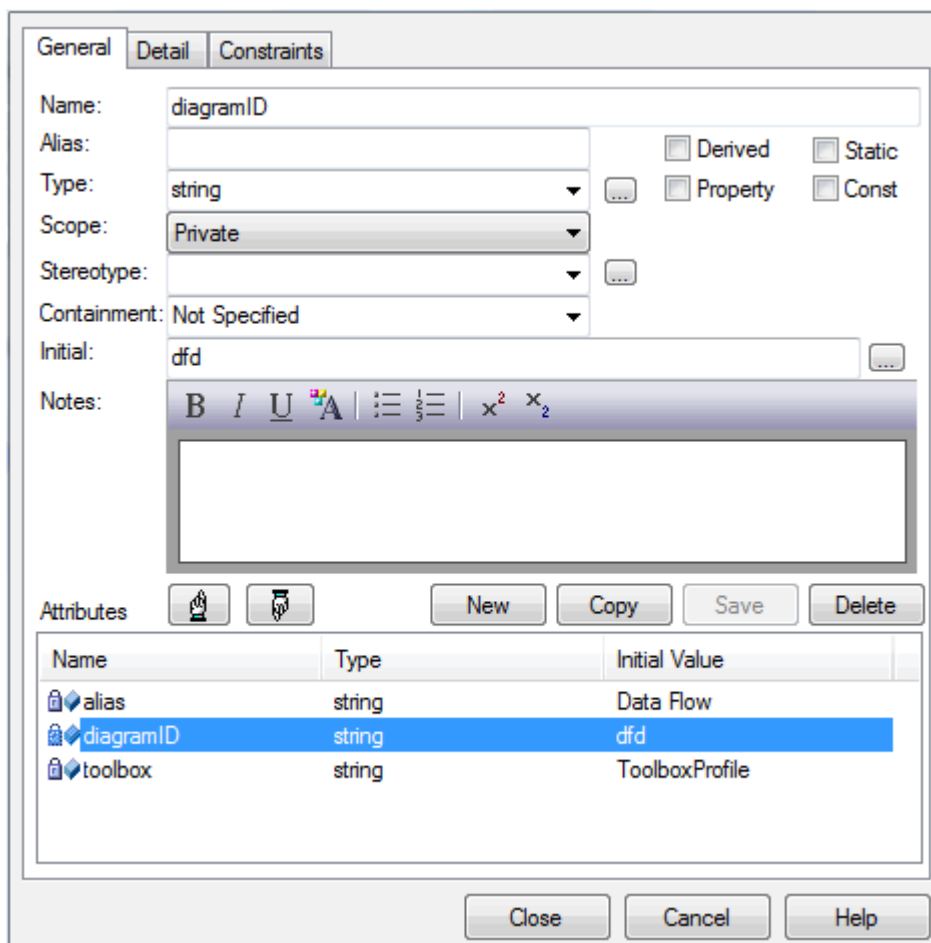
Stereotypes within a UML Profile can have one or more associated Tagged Values. When creating a UML Profile, you define these Tagged Values as attributes of the stereotyped Class.

You can also:

- [Define Stereotype Tags with Predefined Tag Types](#)<sup>[9]</sup>
- [Define Stereotype Tags with Supported Attributes](#)<sup>[9]</sup>
- [Use the Tagged Value Connector](#)<sup>[10]</sup>

To define Tagged Values for a stereotype, follow the steps below:

1. Open the **Attributes** dialog for the stereotyped element.



2. Click on the **New** button to create a new attribute.
3. In the **Name** field, type the name of the stereotype tag.
4. In the **Type** field, click on the drop-down arrow and select the attribute type.
5. In the **Initial** field, type the initial value of the tag. (See the [Add Enumerations to UML Profiles](#)<sup>[13]</sup> topic for the steps for creating enumerated types for Tagged Values.)
6. In the **Notes** field, type a description of the tag.
7. Click on the **Save** button and **Close** button.



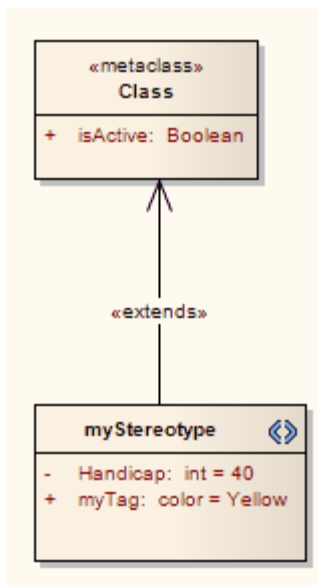
### 1.2.3.1 With Predefined Tag Types

#### Define Predefined Tag Types

To define a stereotype tag with a predefined Tagged Value Type, you must first create the predefined Tagged Value Type. For full instructions on how to do this, see the [Create Structured Tagged Values](#)<sup>[81]</sup> topic.

#### Assign Predefined Tag Types to Stereotypes

To assign a predefined tag type to a stereotype, just create an attribute with the same name. For example, to make the Tagged Value *Handicap* appear in a stereotype, create an attribute named *Handicap*. You can set the default value for the Tagged Value by giving the attribute an *Initial* value.



### 1.2.3.2 With Supported Attributes

Supported stereotype attribute tags are special tags that set the default behavior of stereotyped elements, such as the initial size of the element and the default location of any image files associated with the stereotype. For a list of supported attributes, see the [Supported Attributes](#)<sup>[18]</sup> topic.

To define tags for a stereotype with supported attributes, follow the steps below:

1. Open the **Attributes** dialog for the stereotyped element.

The screenshot shows the 'Create Profile' dialog box with the following configuration:

- Name:** \_sizeX
- Alias:** (empty)
- Type:** int
- Scope:** Private
- Stereotype:** (empty)
- Containment:** Not Specified
- Initial:** 100
- Notes:** Sets size of X

The 'Attributes' section contains the following table:

Name	Type	Initial Value
_sizeX	int	100

- In the **Name** field, type the name of the stereotype tag.
- In the **Initial** field, type the initial value of the tag.

**Note:**

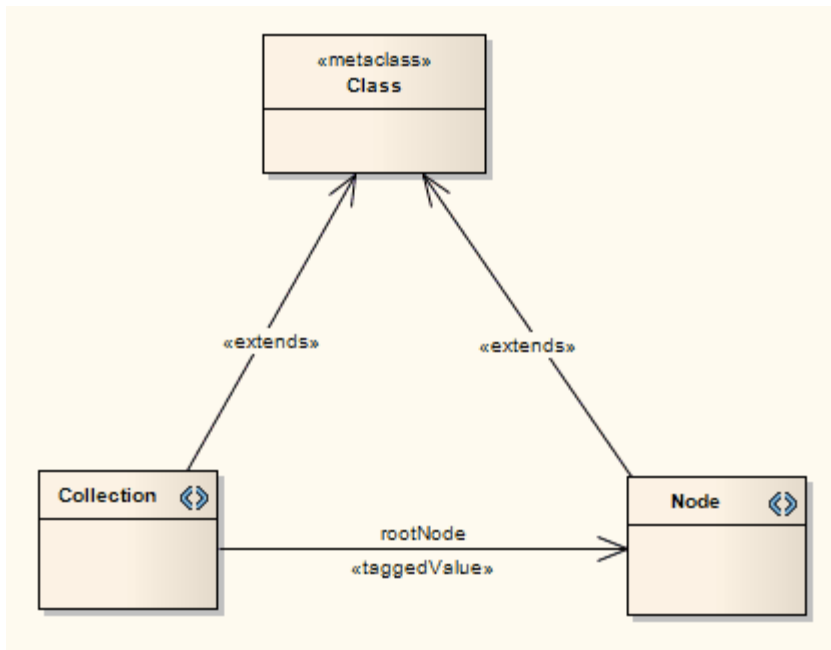
For supported attributes you set only the **Name** (which must match the attributes listed in the supported attributes section) and the **Initial** value; do not set the other values.

- Click on the **Save** button and **Close** button.

### 1.2.3.3 Use the Tagged Value Connector

In a Profile, you can use the *Tagged Value* connector to define a Tagged Value that has as its value the name of an element containing the stereotype pointed to. You select the Tagged Value connector from the **Profile** pages of the Enterprise Architect UML **Toolbox**.

The following diagram demonstrates how you might use the connector. It shows a (*saved* and *imported*) profile that defines two stereotypes: «*Collection*» and «*Node*». The «*Collection*» stereotype has a Tagged Value connector named *rootNode*, pointing to the «*Node*» stereotype.



In the **Tagged Values** window for the connector, against *rootNode*, you click on the selection button ([ ... ]). This displays the **Select <Item>** dialog, which you use to locate elements in the current model with the «Node» stereotype. You can then select one of these elements as the value of the tag. (See the *Work with Elements* section in *UML Modeling With Enterprise Architect - UML Modeling Tool*.)

#### 1.2.4 Define Stereotype Constraints

Defining constraints for stereotypes uses the same procedure as defining constraints for any Class. To define constraints for a stereotype, follow the steps below:

1. Open the Class **Properties** dialog of the stereotype element in a diagram.
2. Click on the **Constraints** tab and click on the **New** button to create a new constraint.

Constraint	Type	Status
t.isRegistered=false	Pre-condition	Approved

Defined Constraints

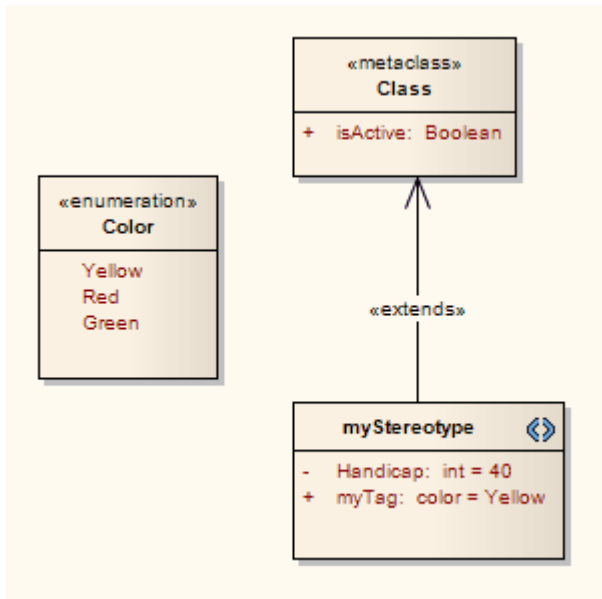
Constraint: t.isRegistered=false Type: Pre-condition Status: Approved

B I U A | :≡ ≡≡ | x<sup>2</sup> x<sub>2</sub>

3. In the **Constraint** field, type the value of the constraint.
4. In the **Type** field, click on the drop-down arrow and select the appropriate type.
5. In the **Status** field, click on the drop-down arrow and select the appropriate status.
6. In the **Notes** field, type any additional information required.
7. Click on the **Save** button, and on the **OK** button to close the dialog.

### 1.2.5 Add Enumeration Elements

Enumerations can be used to restrict the values available to stereotype tags.



#### Note:

Enumerations defined under a Profile Package do not appear as elements in the profile when imported.

To add an Enumeration element, follow the steps below:

1. Open your Profile Package child Class diagram.
2. In the Enterprise Architect UML **Toolbox**, select **More tools | Profile**. The contents of the **Profile** page of the **Toolbox** display.
3. Drag an *Enumeration* item from the toolbox onto the diagram. If the **Properties** dialog does not display, double-click on the element on the diagram.
4. In the **Name** field, type the name of the new Enumeration.
5. Click on the **Details** tab and click on the **Attributes** button. The **Attributes Properties** dialog displays.

The screenshot shows the 'Create Profile' dialog box with the following details:

- General Tab:**
  - Name: Green
  - Alias: (empty)
  - Type: int
  - Scope: Public
  - Stereotype: enum
  - Containment: Not Specified
  - Initial: (empty)
  - Notes: (empty text area with a rich text editor toolbar)
- Properties:**
  - Derived
  - Static
  - Property
  - Const
  - Is Literal
- Attributes Section:**
  - Buttons: New, Copy, Save, Delete
  - Table:

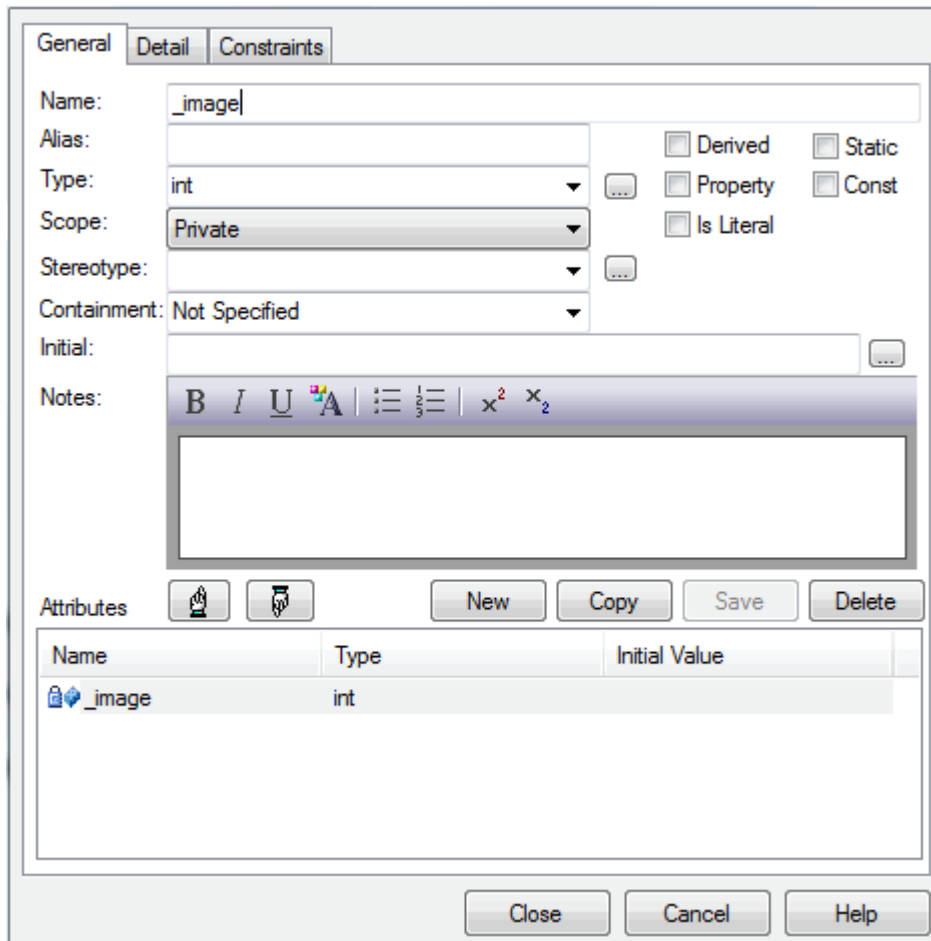
Name	Type	Initial Value
Green	int	
- Bottom Buttons:** Close, Cancel, Help

6. In the **Name** field, type the name of the Enumeration attribute.
  7. In the **Type** field, click on the drop-down arrow and select the appropriate type.
  8. In the **Initial** field, type the initial value of the attribute.
  9. Click on the **Save** button, and repeat steps 6 to 9 for additional attributes.
  10. When you are finished, click on the **Close** button.
  11. Right-click on the *Stereotype* element and select the **Attributes** context menu option. The **Attribute Properties** dialog displays for the stereotype.
  12. In the **Name** field type a name for the attribute.
  13. In the **Type** field type the name of the Enumeration element.
  14. In the **Initial** field type the name of the first enumeration attribute you defined.
  15. Click on the **Save** and **Close** buttons.
- You have now generated a drop-down list for setting the value of the tag in the **Tagged Values** window.

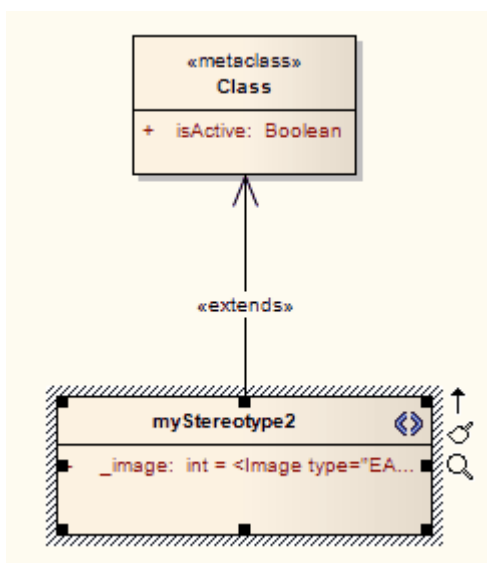
### 1.2.6 Add Shape Scripts

To add a [Shape Script](#) to a stereotype in a UML Profile, follow the steps below:

1. On the Profile Package child diagram, select a *Stereotype* element.
2. Right-click on the element and select the **Attributes** context menu option.
3. In the **Attributes Properties** dialog, in the **Name** field, type `_image`.



4. Click on the [ ... ] button next to the **Initial** field. The **Shape Editor** dialog displays.
  5. Enter the Shape Script in the **Shape Editor** dialog, and click on the **OK** and **Close** buttons.
- The Stereotype element now resembles the example below:



### 1.2.7 Set Default Appearance

You can define the appearance of stereotyped elements and connectors as you [create or edit the stereotypes](#) <sup>13</sup>, using the **Override Appearance** and **Default Colors** panels of the **UML Types** dialog. However, an easier way is to review your completed profile diagram and set the default appearance of the elements and connectors in place.

Simply click on the required element or connector and press **[F4]**, then define the background, font and border colors and border thickness as appropriate, on the **Default Appearance** dialog. (See the *Work with Elements* section in *UML Modeling With Enterprise Architect - UML Modeling Tool*.)

When you [save the profile](#) <sup>16</sup> containing the stereotyped elements and connectors, make sure that you select the **Color and Appearance** checkbox on the **Save UML Profile** dialog.

### 1.2.8 Export a UML Profile

Once you have created a Profile and defined the elements and metaclasses, you can save (export) the Profile to disk for future UML models.

To save a Profile, follow the steps below:

1. If your profile is
  - a single profile spread over multiple diagrams within the same Profile package, find the Profile package in the **Project Browser** window, right-click on it and select the **Save Package as UML Profile** context menu option
  - one of multiple profiles within the same Profile package, right-click anywhere in the background of the Profile diagram and select the **Save as Profile** context menu option
  - a single diagram within the Profile package, choose either the **Save Package as UML Profile** context menu option or the **Save as Profile** context menu option.

#### Note:

The two menu options give slightly different results. See the [Save Profile Options](#) <sup>17</sup> topic.

2. The **Save UML Profile** dialog displays.

The screenshot shows the 'Save UML Profile' dialog box. It contains the following fields and options:

- Profile Name:** ProfileAgate
- Filename:** C:\EA\Profiles\ProfileA.xml (with a browse button)
- Profile Type:**  UML 2.0 (Recommended)  EA Pre-4.0
- Version:** 1.0
- Notes:** (empty text area)
- Include:**
  - Element Size
  - Alternate Image
  - Color and Appearance
  - Code Templates
- Buttons:** Save, Cancel, Help

3. Click on the **[ ... ]** (Browse) button, and select the destination for the XML Profile file to be exported. If necessary, edit the profile filename, but do not delete the .xml extension.
4. Against the **Profile Type**, select the **UML 2.0** radio button.



**Notes:**

- The **UML 2.0** radio button is not available unless the package has the <<profile>> stereotype.
- You can also create stereotypes using the same format as for pre-4.0 versions of Enterprise Architect, by clicking on the **EA Pre-4.0** radio button. This has a reduced feature set and is provided for legacy use.

5. Set the required export options for all stereotypes defined in the profile:
  - **Element Size** - select the checkbox to export the element size attributes
  - **Color and Appearance** <sup>167</sup> - select the checkbox to export the color (background, border and font) and appearance (border thickness) attributes
  - **Alternate Image** - select the checkbox to export the metafile images
  - **Code Templates** - select the checkbox to export the code templates, if they exist.
6. Click on the **Save** button to save the profile to disk.

For information on importing and using the profile in UML modeling, see the *Use Profiles* topic in *Extending UML With Enterprise Architect*.

**1.2.8.1 Save Profile Options**

When you save a UML Profile, you can save it either from the package or from the diagram, depending on whether the Profile is:

- a single profile spread over multiple diagrams within the same Profile package (find the Profile *package* in the **Project Browser**, right-click on it and select the **Save Package as UML Profile** context menu option), which is typically the case for a stereotypes profile
- one of multiple profiles within the same Profile package (right-click anywhere in the background of the Profile *diagram* and select the **Save as Profile** context menu option); for example, when creating multiple toolbox profiles
- a single diagram within the Profile Package (choose *either* the **Save Package as UML Profile** context menu option *or* the **Save as Profile** context menu option).

The two context menu options produce slightly different results. You should take these into consideration, especially in the third instance where you could choose either option.

Save From Diagram	Save From Package	Notes
The profile takes the diagram name.	The profile takes the package name.	Package and diagram names are not necessarily the same, although you can save a lot of confusion if you make them the same or very similar. For example: package <i>GL</i> with diagrams <i>GL1</i> , <i>GL2</i> , <i>GL3</i> .
The profile takes the diagram's notes.	The profile takes the package's notes.	
You can take the default size and appearance (including alternate image) from the diagram object.	You cannot take the default size and appearance from the diagram object.  You can use the <i>_sizeX</i> , <i>_sizeY</i> and <i>_image</i> properties, but there is no equivalent for default colors.	
Can be much faster.	Can be much slower.	Because diagram objects are kept in memory and <b>Project Browser</b> elements aren't.  This is only likely to be an issue if the profile is a large one and you are using a slow network connection to a remote repository.

## 1.2.9 Supported Attributes

### Supported Stereotype Attributes in UML Profiles

The following attributes can be applied to stereotypes in UML Profiles:

Attribute	Meaning
<b>_bezier</b>	Denotes that the line style for a connector is Bezier style. The initial value of the attribute should be set to <b>1</b> .
<b>icon</b>	Contains the path to a bitmap file to be used as the <b>Project Browser</b> icon for all elements with the given stereotype. The bitmap must be 16x16 pixels. For a transparent background, use light grey - RGB(192,192,192).
<b>_image</b>	Shape script definition.
<b>_instanceMode</b>	Used for <a href="#">defining behavior on creating an instance</a> <sup>[20]</sup> .
<b>_instanceOwner</b>	Used for <a href="#">defining behavior on creating an instance</a> <sup>[20]</sup> .
<b>_instanceType</b>	Used for <a href="#">defining behavior on creating an instance</a> <sup>[20]</sup> .
<b>_metatype</b>	Used for <a href="#">defining stereotypes as metatypes</a> <sup>[19]</sup> .
<b>_sizeY</b>	Initial height of the element, in pixels at 100% zoom.
<b>_sizeX</b>	Initial width of the element, in pixels at 100% zoom.
<b>_strictness</b>	Used for <a href="#">restricting application of multiple stereotypes</a> <sup>[19]</sup> .
<b>_treeStyle</b>	Denotes that the line style for a connector is Tree style. The value of the attribute can be set to <b>H</b> (horizontal), <b>V</b> (vertical), <b>LH</b> (lateral horizontal) or <b>LV</b> (lateral vertical).

### Supported Metatype Attributes in UML Profiles

The following attributes can be applied to metatype Classes in UML Profiles, and refer to the stereotypes that extend them:

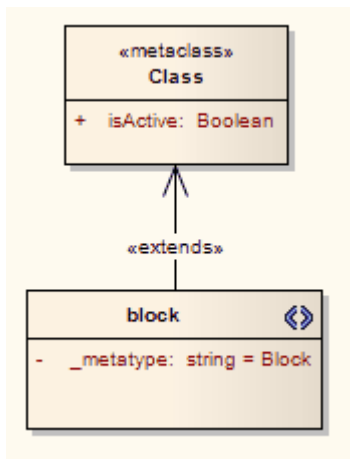
Attribute	Meaning
<b>_AttInh</b>	If set to <b>1</b> , switches on the <i>Inherited Features: Show Attributes</i> setting.
<b>_AttPkg</b>	If set to <b>1</b> , switches on the <i>Attribute Visibility: Package</i> setting.
<b>_AttPri</b>	If set to <b>1</b> , switches on the <i>Attribute Visibility: Private</i> setting.
<b>_AttPro</b>	If set to <b>1</b> , switches on the <i>Attribute Visibility: Protected</i> setting.
<b>_AttPub</b>	If set to <b>1</b> , switches on the <i>Attribute Visibility: Public</i> setting.
<b>_ConInh</b>	If set to <b>1</b> , switches on the <i>Show Element Compartments: Inherited Constraints</i> setting.
<b>_Constraint</b>	If set to <b>1</b> , switches on the <i>Show Element Compartments: Constraints</i> setting.
<b>_DefaultDiagramType</b>	Used for <a href="#">defining child diagram types</a> <sup>[21]</sup> .
<b>_HideStyle</b>	If set to a comma-separated list of stereotypes, sets the <i>Hide Stereotyped Features</i> filter.
<b>_MakeComposite</b>	Used for <a href="#">creating composite elements</a> <sup>[21]</sup> .
<b>_OpInh</b>	If set to <b>1</b> , switches on the <i>Inherited Features: Show Operations</i> setting.
<b>_OpPkg</b>	If set to <b>1</b> , switches on the <i>Operation Visibility: Package</i> setting.

Attribute	Meaning
<b>_OpPri</b>	If set to 1, switches on the <i>Operation Visibility: Private</i> setting.
<b>_OpPro</b>	If set to 1, switches on the <i>Operation Visibility: Protected</i> setting.
<b>_OpPub</b>	If set to 1, switches on the <i>Operation Visibility: Public</i> setting.
<b>_PType</b>	If set to 1, switches on the <i>Show element type (Port and Part only)</i> setting.
<b>_ResInh</b>	If set to 1, switches on the <i>Show Element Compartments: Inherited Responsibilities</i> setting.
<b>_Responsibility</b>	If set to 1, switches on the <i>Show Element Compartments: Responsibilities</i> setting.
<b>_Runstate</b>	If set to 1, switches on the <i>Hide Object Runstate in current diagram</i> setting.
<b>_Tag</b>	If set to 1, switches on the <i>Show Element Compartments: Tags</i> setting.
<b>_TagInh</b>	If set to 1, switches on the <i>Show Element Compartments: Inherited Tags</i> setting.

### 1.2.9.1 Define a Stereotype as a Metatype

The **\_metatype** attribute is applied to a stereotype element. This is used where users want to hide the identity of an element as a stereotyped UML element. It is also a method of getting custom types to appear in contexts where only Enterprise Architect's inbuilt types would normally appear; for example in the lists of element types in the [Relationship Matrix](#).

In the following example from SysML, *block* is defined as a stereotype that extends a UML Class.



However, a SysML user isn't interested in UML Classes, only in SysML Blocks. An element created from a stereotype defined this way, while behaving like a stereotyped Class in most contexts:

- Shows *Block Properties* rather than *Class Properties* as the title of its **Properties** dialog
- Is auto-numbered as *Block1* not *Class1* on creation, and
- Appears as *Block* not *Class* in many other contexts throughout Enterprise Architect.

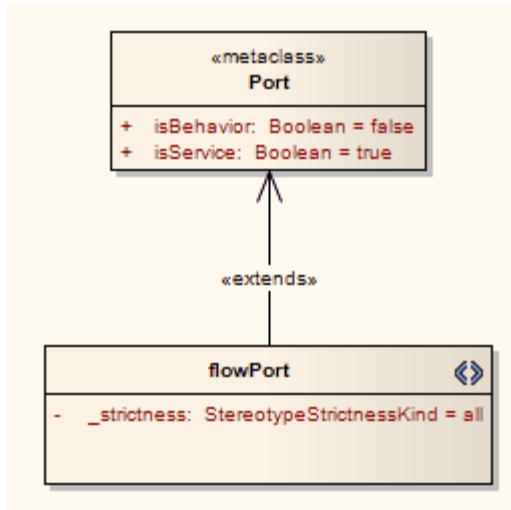
### 1.2.9.2 Define Multiple-Stereotype Level

The **\_strictness** attribute is applied to a stereotype element. It defines to what level multiple stereotypes can be applied to an element. The type of the attribute is *StereotypeStrictnessKind* and it can have one of four values:

- *profile*, which states that an element of this type cannot be given more than one different stereotype from the same profile
- *technology*, which states that an element of this type cannot be given more than one different stereotype from the same technology

- *all*, which states that an element of this type cannot have multiple stereotypes at all, or
- *none*, which is the default Enterprise Architect behaviour and states that there are no restrictions on the use of multiple stereotypes.

The following example is from SysML and shows that a «*flowPort*» cannot have any other stereotype applied to it.



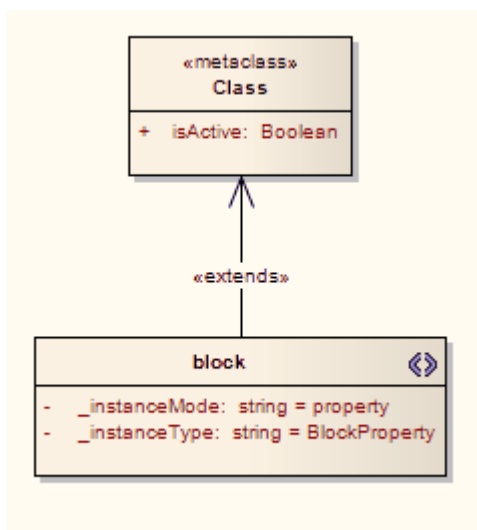
### 1.2.9.3 Define Creation of Instance

The **\_instanceType** attribute is applied to a stereotype element and defines what kind of element is created as an instance of this element type. The value corresponds to the metatype given to a stereotype using the **\_metatype** attribute. It is shown on the **Paste Element** dialog (see the *Work with Diagrams* section in *UML Modeling With Enterprise Architect - UML Modeling Tool*) and is translated if it matches an Enterprise Architect element type.

The **\_instanceMode** attribute is applied to a stereotype element and controls the text in the **Paste Element** dialog after being translated. Valid values are **instance** and **property**, with the default being **instance**.

The **\_instanceOwner** attribute is applied to a stereotype element and controls the text in the **Paste Element** dialog. It is translated if it matches an Enterprise Architect element type. The default value is **Element**.

The following example from SysML shows that when an instance of a Block is created, it is created as a **BlockProperty** element.



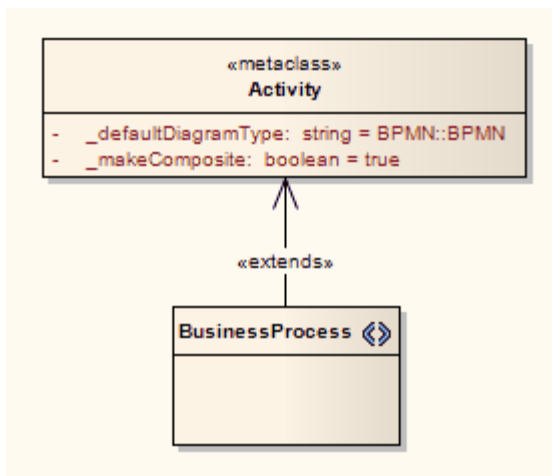
### 1.2.9.4 Create Composite Elements

The **\_makeComposite** attribute is applied to a metaclass element, not a stereotype element. It defines whether an element is always made composite when created.

#### Notes:

- A stereotyped package is not by default created with a child diagram, so you should use the **\_makeComposite** attribute to ensure the child diagram is created.
- Unless you also use the **\_defaultDiagramType** attribute to [define the child diagram type](#)<sup>[21]</sup>, the child diagram created is a Package diagram.

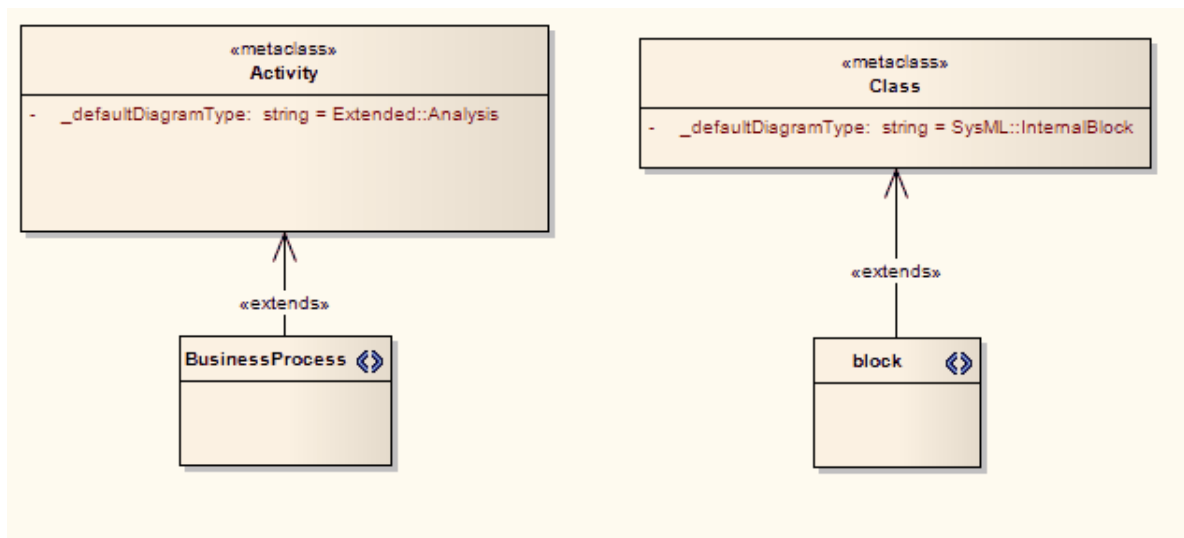
The following example from BPMN shows that a *BusinessProcess* element is always created as a Composite element with a BPMN custom child diagram.



### 1.2.9.5 Define Child Diagram Types

The **\_defaultDiagramType** attribute is applied to a metaclass element, not a stereotype element. It defines the type of diagram created when an element is [made composite](#)<sup>[21]</sup>. This attribute can take as its name any of the inbuilt diagram types of Enterprise Architect (these are listed below). Alternatively, if a custom diagram type is required, it should be prefixed with the diagram profile name and '::`'.`

The following examples show a `«BusinessProcess»Activity` that, when made a composite element, automatically creates an Analysis diagram, and a `«block»stereotype` that creates a SysML *InternalBlock* custom diagram.



You can also use the `_defaultDiagramType` attribute for packages, extending the Package metaclass.

### Values For `_defaultDiagramType`

The following initial values should be used to refer to Enterprise Architect's inbuilt diagram types:

- UML Behavioral::Use Case
- UML Behavioral::Activity
- UML Behavioral::State Machine
- UML Behavioral::Communication
- UML Behavioral::Sequence
- UML Behavioral::Timing
- UML Behavioral::Interaction Overview
- UML Structural::Package
- UML Structural::Class
- UML Structural::Object
- UML Structural::Composite Structure
- UML Structural::Component
- UML Structural::Deployment
- Extended::Custom
- Extended::Requirements
- Extended::Maintenance
- Extended::Analysis
- Extended::User Interface
- Extended::Data Modeling
- Extended::ModelDocument.

#### 1.2.10 Stereotypes Profiles

In Enterprise Architect 6, an MDG Technology could consist of many UML Profiles, each representing an Enterprise Architect UML **Toolbox** page. Each profile could include stereotype definitions alongside redefinitions of standard UML elements. This meant that to define a stereotype in a technology, it had to appear in one of the **Toolbox** pages.

Enterprise Architect 7 takes a different approach, splitting the task of defining the stereotypes and the task of defining the **Toolbox** pages into separate profiles. The MDG Technology's stereotypes are contained in one or more profiles. The stereotypes within each profile use the profile name as the namespace.

Create one or more packages with the `«profile»` stereotype, each package name being the namespace. Within

each package create profile diagrams defining all the stereotypes in the namespace. You can use multiple diagrams to do this, but do not use nested packages.

Give the «profile» package a description in the **Notes** field (eg *MDG Technology for BPMN*). When all of the stereotypes are defined (make sure that every stereotype extends at least one *Metaclass*) right-click on the profile package in the **Project Browser** and select the **Save Package as UML Profile** context menu option, then proceed as usual.

## 1.3 Quick Linker

### Introduction

The Quick Linker provides a simple and fast way to create new elements and connectors on a diagram. When an element is selected in a diagram, the Quick Linker icon is displayed in the upper right corner of a element. Simply clicking and dragging the icon enables you to create new connectors and elements. The philosophy behind the built-in Quick Linker definitions has always been to provide not a complete list of valid or legal connections, but a short and convenient list of the commonest connections for the given context. As part of a UML Profile, you can add to or replace the built-in Quick Linker definitions; the following sections of this document explain how:

- [Quick Linker Definition Format](#) <sup>[23]</sup>
- [Quick Linker Example](#) <sup>[25]</sup>
- [Hide Default Quick Linker Settings](#) <sup>[26]</sup>
- [Quick Linker Object Names](#) <sup>[27]</sup>.

### Customized Quick Linker Settings

A Quick Linker definition is a Comma Separated Value (CSV) format file. It is best manipulated in a spreadsheet which should be set up to save the CSV file as comma-separated text without quotation marks around text fields.

To add a Quick Linker definition file to a profile or technology, simply place a *DocumentArtifact* element onto the *Profile* diagram. Give it the name 'QuickLink' then double-click on it. Open your CSV file in a text editor such as Notepad and copy and paste the contents into the *DocumentArtifact* element. The definitions are saved with the profile and are processed and applied when the profile is imported. The same applies if a profile is included within a technology, with the proviso that the QuickLink element must be in the same profile as the link stereotype definitions. This means that a technology could have a set of Quick Link definitions for each profile.

#### 1.3.1 Quick Linker Definition Format

A Quick Linker definition is a text file consisting of records terminated by new-line characters. Each record must consist of 23 comma-separated fields, as defined by the table below. The values of each field must not be in quotes (" "). A Quick Linker definition can include comments: all lines that begin with // are ignored by Enterprise Architect.

Each record of the Quick Linker definition represents a single entry on the Quick Linker menu. Some fields define the menu command; some fields can be thought of as filters, with the entry being ignored if the filter condition isn't met.

A Quick Linker definition has the following fields.

Column	Field	Description
A	Source Element Type	The row is ignored unless a connector is being dragged away from this type of element.
B	Source Stereotype Filter	If set, the row is ignored unless a connector is being dragged away from an element with this stereotype.
C	Target Element Type	If set, the row is ignored unless a connector is being dragged onto this type of element.  If blank, the row is ignored unless a connector is being dragged onto an

Column	Field	Description
		empty piece of diagram.
D	Target Stereotype Filter	If set and <b>Target Element Type</b> also set, the row is ignored unless a connector is being dragged onto an element with this stereotype.
E	Diagram Filter	Contains either an inclusive or exclusive list of diagrams, which limits the diagrams the given kind of connector can be included on.  Each diagram name is terminated by a semi-colon. Excluded diagram names are preceded by an exclamation mark.  Example of an inclusive list: <i>Collaboration;Object;Custom;</i>  Example of an Exclusive list: <i>!Sequence;</i>
F	New Element Type	If set and <b>Create Element</b> also set, results in the creation of an element of this type.
G	New Element Stereotype	If set and <b>Create Element</b> also set, results in the creation of an element with this stereotype.
H	New Link Type	If set and <b>Create Link</b> also set, results in the creation of a connector of this type.
I	New Link Stereotype	If set and <b>Create Link</b> also set, results in the creation of a connector with this stereotype.
J	New Link Direction	Can be: <ul style="list-style-type: none"> <li>• directed (always creates an association from source to target)</li> <li>• from (always creates an association from target to source)</li> <li>• undirected (always creates an association with unspecified direction)</li> <li>• bidirectional (always creates a bi-directional association), or</li> <li>• to (creates either a directed or undirected association, depending on the value of the <b>Association Direction</b> option).</li> </ul> <p><b>Note:</b> Not all of the above work with all connector types; for example, you cannot create a bi-directional Generalization.</p>
K	New Link Caption	If a new connector is being created but not a new element, then this is the text that appears on the context menu.
L	New Link & Element Caption	If a new connector AND a new element are being created, then this is the text that appears on the context menu.
M	Create Link	If set to <b>TRUE</b> , results in creation of a new connector; otherwise should be left blank.
N	Create Element	If set to <b>TRUE</b> the row is ignored unless a connector is being dragged onto an empty piece of diagram and results in creation of a new element; otherwise should be left blank.  This overrides the values of <b>Target Element Type</b> and <b>Target Stereotype Filter</b> .
O	Disallow Self connector	Should be set to <b>TRUE</b> if self connectors are invalid for this kind of connector; otherwise should be left blank.
P	Exclusive to ST Filter + No inherit from Metatype	If set to <b>TRUE</b> , indicates that elements of type <i>Source Element Type</i> with the stereotype <i>Source Stereotype Filter</i> do not display the Quick Linker definitions of the equivalent unstereotyped element.



Column	Field	Description
Q	Menu Group	If set, indicates the name of a sub-menu in which a menu item is created.
R	Complexity Level	Not implemented, always set to 0.
S	Target Must Be Parent	If set to <b>TRUE</b> this menu item only appears when dragging from a child element to its parent; for example from a port to its containing Class.
T	Embed element	If set to <b>TRUE</b> the element being created is embedded in the target element; otherwise should be left blank.
U	Precedes Separator LEAF	If set to <b>TRUE</b> results in a menu separator being added to the Quick Linker menu; otherwise should be left blank.
V	Precedes Separator GROUP	If set to <b>TRUE</b> results in a menu separator being added to the Quick Linker sub-menu; otherwise should be left blank.
W	Dummy Column	Depending on which spreadsheet application you use, this column might require a value in every cell to force CSV export to work correctly with trailing blank values.

### 1.3.2 Quick Linker Example

This example uses a Class element with the stereotype «*quick*». The example scenario is this: when you drag a connector away from one of these elements, you want to create a Dependency either to or from a component element. When you drag a connector onto an existing *Port* or component element, you want a Dependency either to or from the component **or**, in the case of a component, you want to be able to create an embedded Port element.

This results in 8 records in the [Quick Linker definition](#) [23] file.

1. Dependency to new Component
2. Dependency from new Component
3. Dependency to existing Component
4. Dependency from existing Component
5. Dependency to existing Port
6. Dependency from existing Port
7. Dependency to existing Component, create new Port
8. Dependency from existing Component, create new Port

In the spreadsheet, this is implemented by the following values:

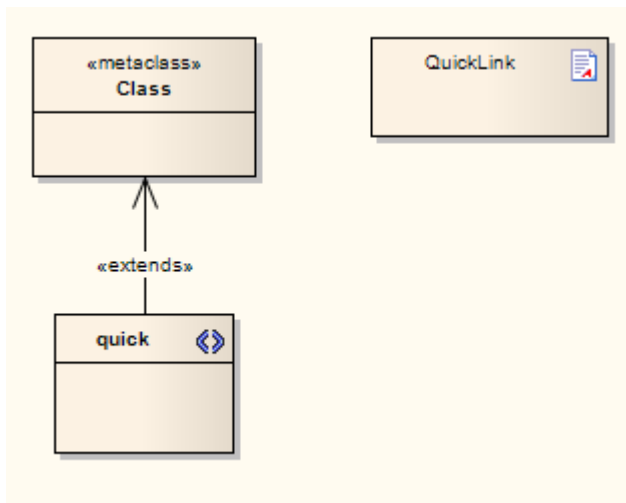
	A	B	C	E	F	H	J	K
1	//Source Element Type	Source ST filter	Target Element Type	Diagram Filter	New Element Type	New Link Type	New Link Direction	New Link Caption
2	Class	quick			Component	Dependency	to	
3	Class	quick			Component	Dependency	from	
4	Class	quick	Component			Dependency	to	Dependency to
5	Class	quick	Component			Dependency	from	Dependency from
6	Class	quick	Port			Dependency	to	Dependency to
7	Class	quick	Port			Dependency	from	Dependency from
8	Class	quick	Component		Port	Dependency	to	
9	Class	quick	Component		Port	Dependency	from	

	L	M	N	O	P	Q	R	S	T	U	V
1	New Link & Element Caption	Create Link	Create Element	Disallow Self connector	No inherit from metatype	Menu Group	Complexity Level	Target Must Be Parent	Embed element	Precedes Separator LEAF	Precedes Separator GROUP
2	Dependency to	TRUE	TRUE	TRUE	TRUE	Component	0				
3	Dependency from	TRUE	TRUE	TRUE	TRUE	Component	0			TRUE	
4		TRUE		TRUE	TRUE		0				
5		TRUE		TRUE	TRUE		0			TRUE	
6		TRUE		TRUE	TRUE		0				
7		TRUE		TRUE	TRUE		0			TRUE	
8	Dependency to	TRUE	TRUE	TRUE	TRUE	Port	0		TRUE		
9	Dependency from	TRUE	TRUE	TRUE	TRUE	Port	0		TRUE	TRUE	

This saves to the following CSV:

```
Class,quick,,,,Component,,Dependency,,to,,Dependency to,TRUE,TRUE,TRUE,TRUE,Component,0,,,,
Class,quick,,,,Component,,Dependency,,from,,Dependency from,TRUE,TRUE,TRUE,TRUE,Component,0,,,TRUE,,
Class,quick,Component,,,,Dependency,,to,Dependency to,,TRUE,,TRUE,TRUE,,0,,,,
Class,quick,Component,,,,Dependency,,from,Dependency from,,TRUE,,TRUE,TRUE,,0,,,TRUE,,
Class,quick,Port,,,,Dependency,,to,Dependency to,,TRUE,,TRUE,TRUE,,0,,,,
Class,quick,Port,,,,Dependency,,from,Dependency from,,TRUE,,TRUE,TRUE,,0,,,TRUE,,
Class,quick,Component,,Port,,Dependency,,to,Dependency to,TRUE,TRUE,TRUE,TRUE,Port,0,,TRUE,,,
Class,quick,Component,,Port,,Dependency,,from,Dependency from,TRUE,TRUE,TRUE,TRUE,Port,0,,TRUE,TRUE,,
```

You can create the following profile and cut and paste the CSV data into the document artifact to test the effect.



### 1.3.3 Hide Default Quick Linker Settings

If you have a Quick Linker definition with the *Exclusive to stereotype* flag (column P) set to **TRUE**, then the default Quick Linker definitions between the given source and target are overridden. However, you might want to override the defaults without actually having a Quick Linker definition. For example, if you don't define any Quick Links for a «quick» Class to another «quick» Class, Enterprise Architect displays the default Quick Links for a Class to another Class. To override this behaviour, create a Quick Linker definition that has the source element type, source stereotype filter, target element type and target stereotype filter fields (columns A, B, C and D) all set, with the **Exclusive to stereotype** flag (column P) set to **TRUE**, and with the new link type field (column H) set to **<none>**.

For example, add this line to the example in [Quick Linker Example](#) <sup>25</sup>:

```
Class,quick,Interface,,,,<none>,,,,,TRUE,,0,,,,
```

This overrides the default Class-to-Interface Quick Links when a Quick Link is dragged from a «quick» Class to an Interface element.

**Note:**

This technique does not affect the automatic appearance of Dependency, Trace, Information Flow and Help items on the Quick Linker menu.

### 1.3.4 Quick Linker Object Names

#### List of Element Types

The following element names can be used in Quick Linker definitions:

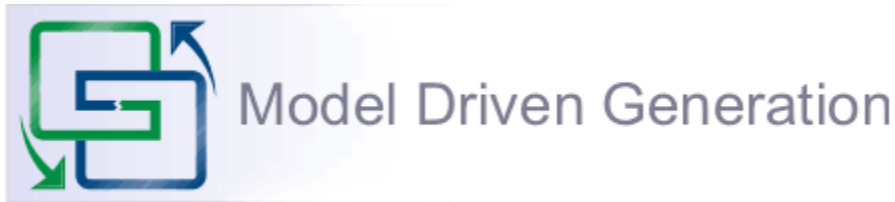
Action	ExecutionEnvironment	ObjectNode
ActionPin	ExitPoint	Package
Activity	ExitState	Part
ActivityParameter	ExpansionNode	Port
ActivityPartition	ExpansionRegion	ProvidedInterface
Actor	Feature	Receive
Artifact	GUIElement	RequiredInterface
Boundary	HistoryState	Requirement
CentralBufferNode	InformationItem	Screen
Change	InitialActivity	Send
ChoiceState	InitialState	Sequence
Class	InteractionOccurrence	Signal
Collaboration	Interface	State
Component	Issue	StateLifeline
Decision	InterruptableActivityRegion	StateMachine
DeepHistoryState	JunctionState	Synchronization_H
Deployment Specification	MergeNode	Synchronization_V
Device	MessageEndpoint	SynchState
DiagramGate	n-ary Association	UMLDiagram
EntryPoint	Node	UseCase
EntryState	Object	ValueLifeline

#### List of Connector Types

The following connector names can be used in Quick Linker definitions:

Aggregation	Deployment	PackageMerge
Association	Extension	Realization
CommunicationPath	Generalization	Redefinition
Composition	InterfaceLink	Sequence
ConnectorLink	Manifest	StateFlow
ControlFlow	Nesting	UCExtends
DelegateLink	ObjectFlow	UCIncludes
Dependency	PackageImport	UseCase

## 2 MDG Technologies in SDK



The Model Driven Generation (MDG) Technologies enable Enterprise Architect users to access and use resources pertaining to a specific technology in Enterprise Architect. There are various options for an administrator or individual user to bring MDG Technologies into use with Enterprise Architect, as described in *Extending UML With Enterprise Architect*. You should read the MDG Technology topics to understand how MDG Technologies are accessed and used within Enterprise Architect, especially the *Manage MDG Technologies* topic.

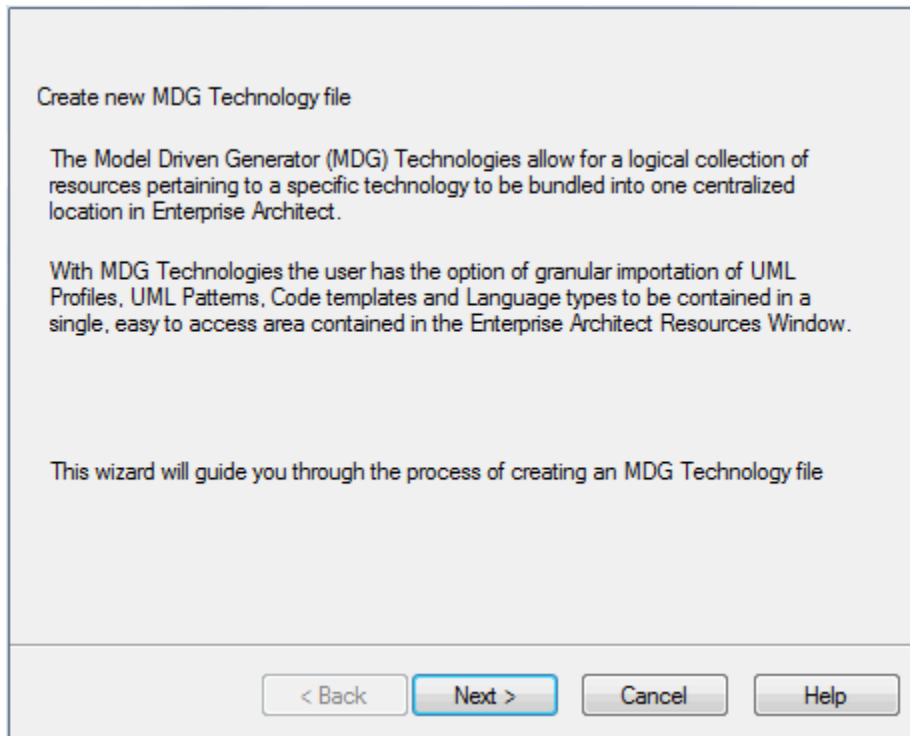
A further option is that Technology Developers can develop new MDG Technologies and deploy them to the project team as appropriate; this is described in the following topics:

- [Create MDG Technologies](#) <sup>[28]</sup>
- [Working With MTS Files](#) <sup>[44]</sup>
- [Customize Toolbox Profiles](#) <sup>[45]</sup>
- [Create Diagram Profiles](#) <sup>[50]</sup>
- [Create Tasks Pane Profiles](#) <sup>[52]</sup>
- [Define Validation Configuration](#) <sup>[56]</sup>
- [Incorporate Model Templates](#) <sup>[57]</sup>
- [Deploy an MDG Technology](#) <sup>[57]</sup>

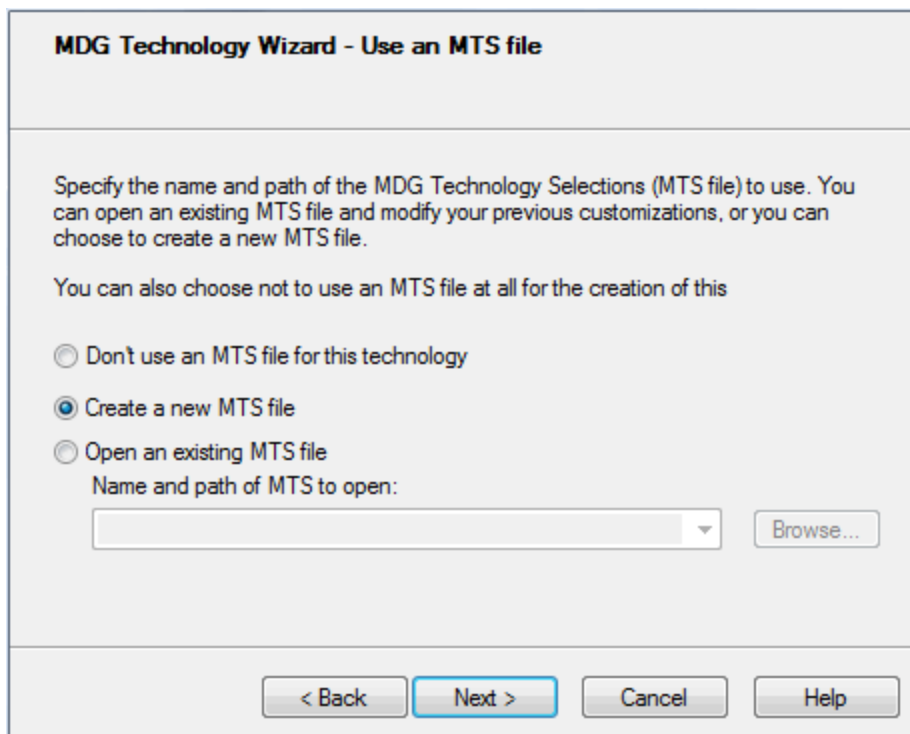
### 2.1 Create MDG Technologies

Using the MDG Technology Wizard, you can create MDG Technology files that can include UML Profiles, code modules, Patterns, images, Tagged Value Types, RTF report templates, linked document templates, **Toolbox** pages and **Task Pane** pages. To create an MDG Technology file, follow the steps below:

1. Select the **Tools | Generate MDG Technology File** menu option. The **MDG Technology Creation Wizard** screen displays.



2. Click on the **Next** button to proceed. The **MDG Technology Wizard** prompts you to:
  - Create an MDG Technology File by creating a new MDG Technology Selection (MTS) file
  - Create an MDG Technology File using an existing MTS file
  - Not use any MTS file.



(An MTS file stores the selected options that you define during the creation of an MDG Technology File. If you use an MTS file, you can later modify it to add or remove specific items in the MDG Technology File. This is the recommended process.)

3. Select the appropriate MTS file option. Click on the **Next** button.

If you selected an MTS file, the **MDG Technology Wizard** prompts you to save the changes in the existing MTS file or into a new MTS file. This enables you to create a modification based on the existing MTS file, while preserving the original file.

The screenshot shows a dialog box titled "MDG Technology Wizard - Save MTS file". The main text reads: "You can save your changes in the MTS file that you opened, or enter the name and path of a new MTS file." Below this, there is a label "Name and path of MTS to save:" followed by a text input field containing "Y:\Dev\Roy\MY\_MTS.mts" and a "Browse..." button. At the bottom, there are four buttons: "< Back", "Next >", "Cancel", and "Help".

4. If necessary, type in or browse for the required file path and name. Click on the **Next** button. The **MDG Technology Wizard - Create** screen displays.

The screenshot shows a dialog box titled "MDG Technology Wizard - Create" with the subtitle "Please specify the technology to be created". The form contains several fields: "Technology:" with the value "MDG Technology For Workflows"; "Filename:" with the value "roy\UML Profiles\WKFL\MDG Technology For Workflows.xml" and a browse button; "ID:" with the value "WKFL" and "Version:" with the value "0.1"; "Icon:" with the value "Y:\Dev\Help\Source\EA Next\Help\_Graphics\uml\_profiles.bn" and a browse button; "Logo:" with the value "Y:\Dev\Help\Source\EA Next\Help\_Graphics\Banner1.BMP" and a browse button; "URL:" and "Support:" fields; and a "Notes:" text area containing "Technology to generate and collate project development workflows." At the bottom, there are four buttons: "< Back", "Next >", "Cancel", and "Help".

5. Complete the fields on this screen as follows:

Option	Use to
<b>Technology</b>	Type the name of the MDG Technology.
<b>Filename</b>	Type or select the path and filename of the MDG Technology File (the file extension for this file is .xml).
<b>ID</b>	Type a reference for the MDG Technology File, up to 12 characters long.
<b>Version</b>	Type the version number of the MDG Technology File.
<b>Icon</b>	(Optional) Type or select the path and file name of the graphics file containing the technology icon. The icon is a 16x16 bitmap image that is shown in the list of technologies on the left of the <b>MDG Technologies</b> dialog.
<b>Logo</b>	(Optional) Type or select the path and file name of the graphics file containing the technology logo. The logo is a 64x64 bitmap image that is shown in the display pane on the top-right corner of the <b>MDG Technologies</b> dialog.
<b>URL</b>	(Optional) If you have any website product information that might be helpful for users of this Technology, type or paste the URL in this field.
<b>Support</b>	(Optional) If you have any web-based or other support facility that might be helpful for users of this Technology, type or paste the contact address in this field.
<b>Notes</b>	Type a short explanation of the functionality of the MDG Technology.

6. Click on the **Next** button. The **MDG Technology Wizard - Contents** screen displays.

**MDG Technology Wizard - Contents**  
Select the information to be included in your technology

**Metamodel**

- Profiles
- Patterns
- Diagram Types
- Toolboxes
- Taskpages
- Tagged Value Types

**Code**

- Code Modules
- MDA Transforms

**Reports**

- RTF Templates
- Linked Document Templates

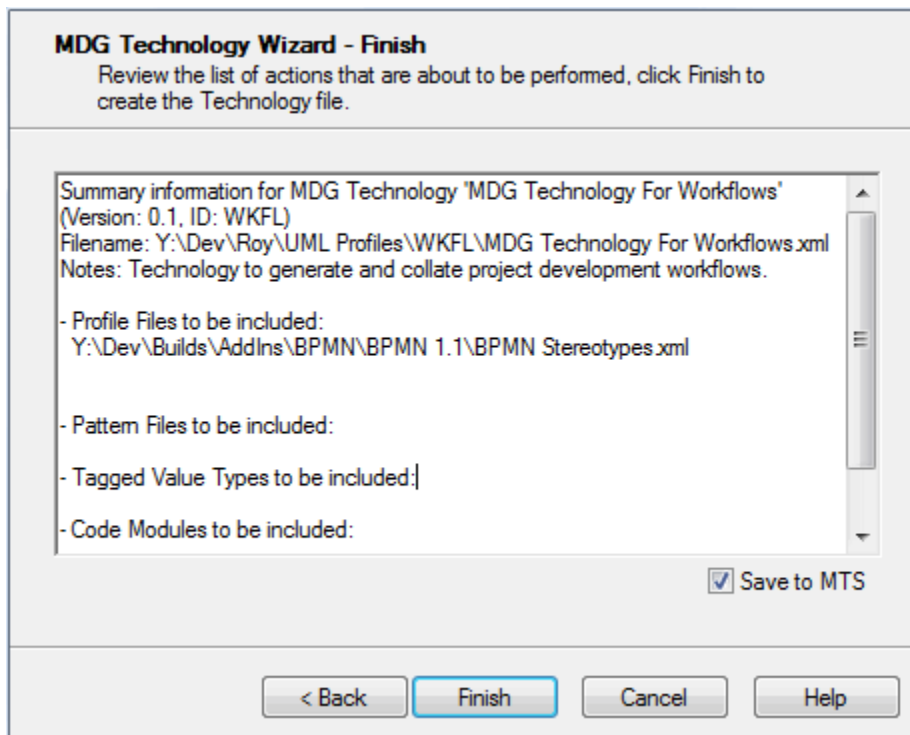
**Other**

- Images
- Scripts

< Back    Next >    Cancel    Help

7. Select the checkbox for each item to be included in the MDG Technology file. Each selection runs specific dialogs to enable definition of the specific items to be included in the MDG Technology, as described in the following topics:

- [Add a Profile](#) <sup>[33]</sup>
  - [Add a Pattern](#) <sup>[33]</sup>
  - [Add a Diagram Profile](#) <sup>[34]</sup>
  - [Add a Toolbox Profile](#) <sup>[35]</sup>
  - [Add Task Panel Pages](#) <sup>[36]</sup>
  - [Add Tagged Value Types](#) <sup>[37]</sup>
  - [Add Code Modules](#) <sup>[38]</sup>
  - [Add MDA Transformations](#) <sup>[40]</sup>
  - [RTF Report Templates](#) <sup>[42]</sup>
  - [Linked Document Templates](#) <sup>[43]</sup>
  - [Add Images](#) <sup>[41]</sup>
  - [Add Scripts](#) <sup>[41]</sup> (Corporate and 'Suite' editions).
8. Work through the dialogs displayed in response to your choices, and when all are complete, click on the **Next** button. The **MDG Technology Wizard - Finish** screen displays, providing information on the items included in the MDG Technology File.



9. If you have used an MTS file and want to update it, select the **Save to MTS** checkbox.
10. If you are satisfied with the selection of items, click on the **Finish** button.
- You can now [edit the MTS file](#) <sup>[44]</sup>, if required, to add further items such as:

- Model Search definitions
- Model Views
- Model Validation configurations
- Model Templates.

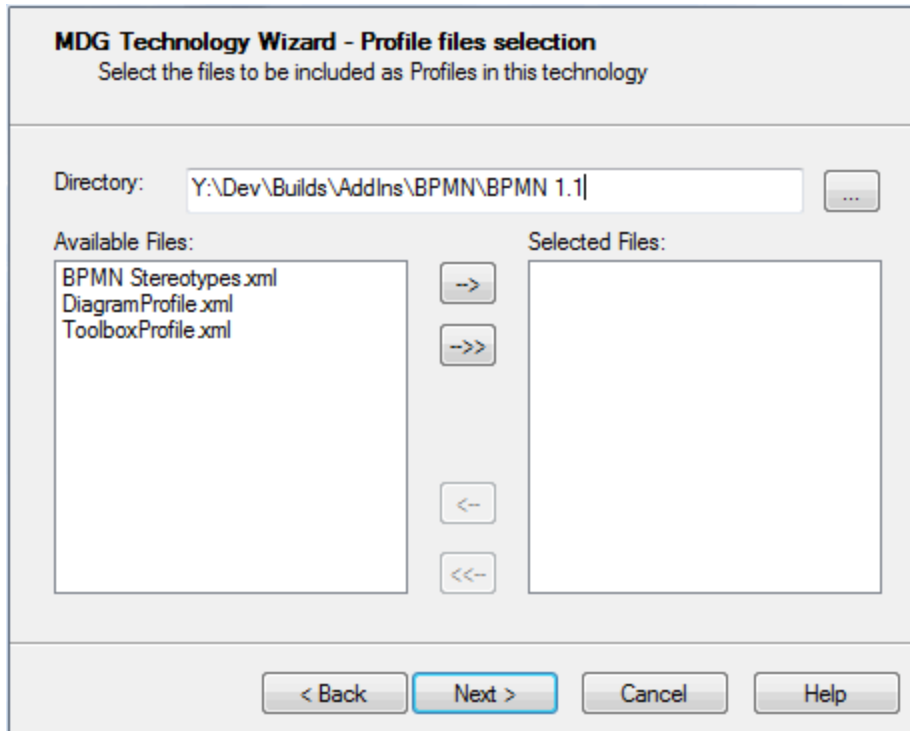
To make the MDG Technology File accessible to an Enterprise Architect model, you must *add* the technology file path to the **MDG Technologies - Advanced** dialog. See the *Access Remote MDG Technologies* topic in *Extending UML With Enterprise Architect*.



### 2.1.1 Add a Profile

When creating an MDG Technology file, you can include UML 2.1-compliant profiles [that you have defined](#)<sup>[5]</sup>. To use the Profiles section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the Profiles checkbox. The [MDG Technology Wizard - Profile files selection](#) dialog displays.



2. In the **Directory** field, navigate to the directory containing the required Profile or Profiles. The Profile files are automatically listed in the **Available Files** panel.
3. To select each required Profile individually, highlight the Profile in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available Profiles, click on the -->> button.

#### Notes:

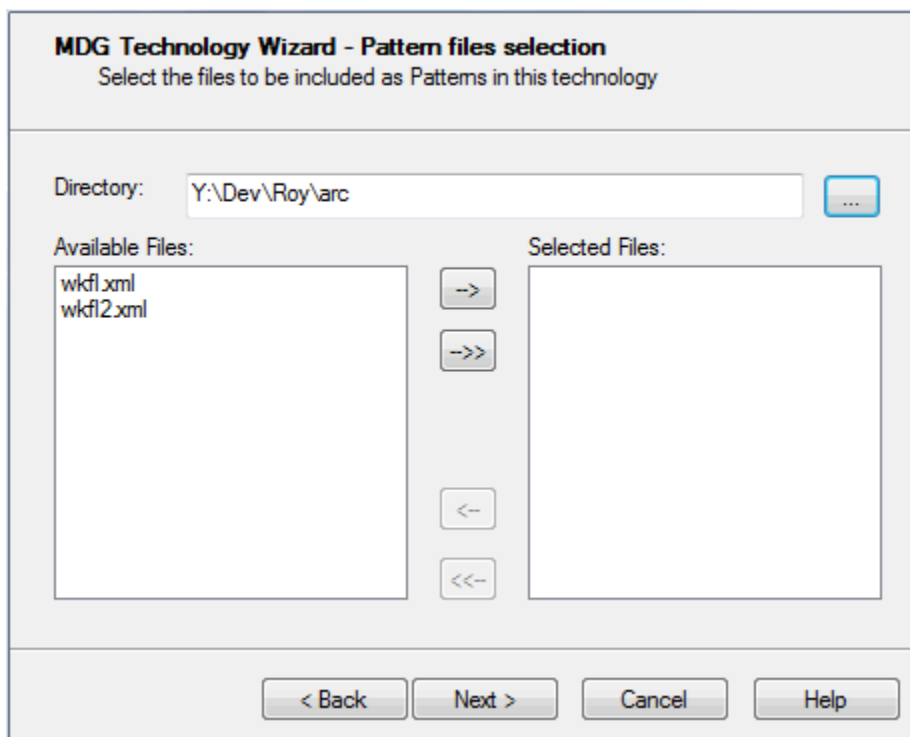
- DO NOT select diagram profiles or toolbox profiles on this dialog; this would generate conflicting commands in the .MTS file.
- Make sure you do include your [stereotypes profile](#)<sup>[22]</sup>.

4. Click on the **Next** button to proceed.

### 2.1.2 Add a Pattern

When creating an MDG Technology file, you can include Patterns. To use the Patterns section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Patterns** checkbox. The [MDG Technology Wizard - Pattern files selection](#) dialog displays.

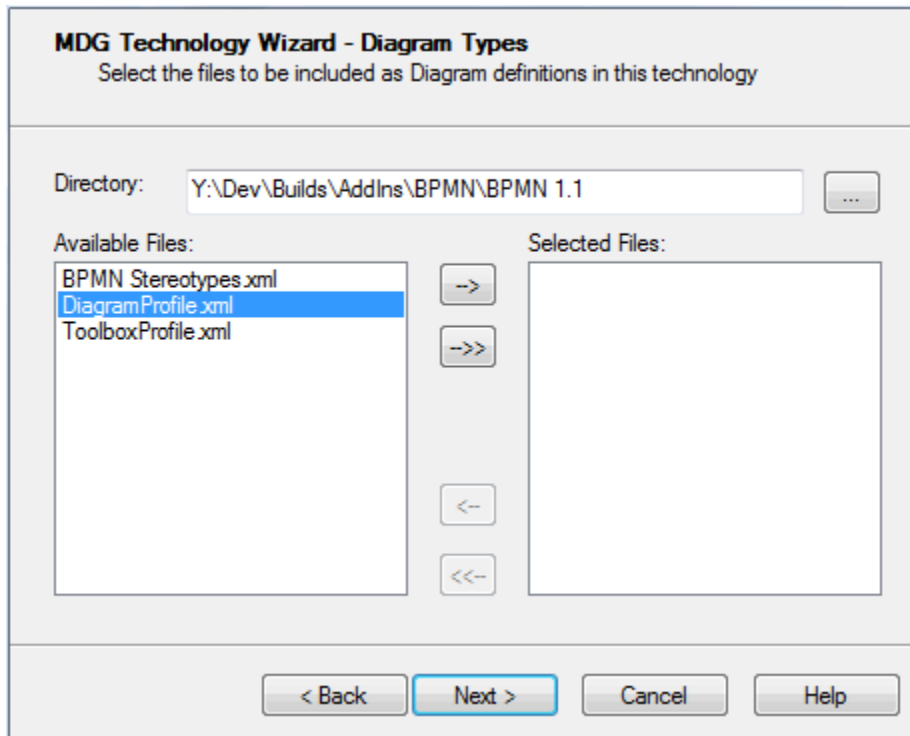


2. In the **Directory** field, navigate to the directory containing the required Pattern or Patterns. The Pattern files are automatically listed in the **Available Files** panel.
3. To select each required Pattern individually, highlight the Pattern in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available Patterns, click on the -->> button.
4. Click on the **Next** button to proceed.

### 2.1.3 Add a Diagram Profile

When creating an MDG Technology file, you can include a diagram profile [that you have defined](#)<sup>[50]</sup>. To use the diagram profiles section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Diagram Types** checkbox. The **MDG Technology Wizard - Diagram Types** dialog displays.

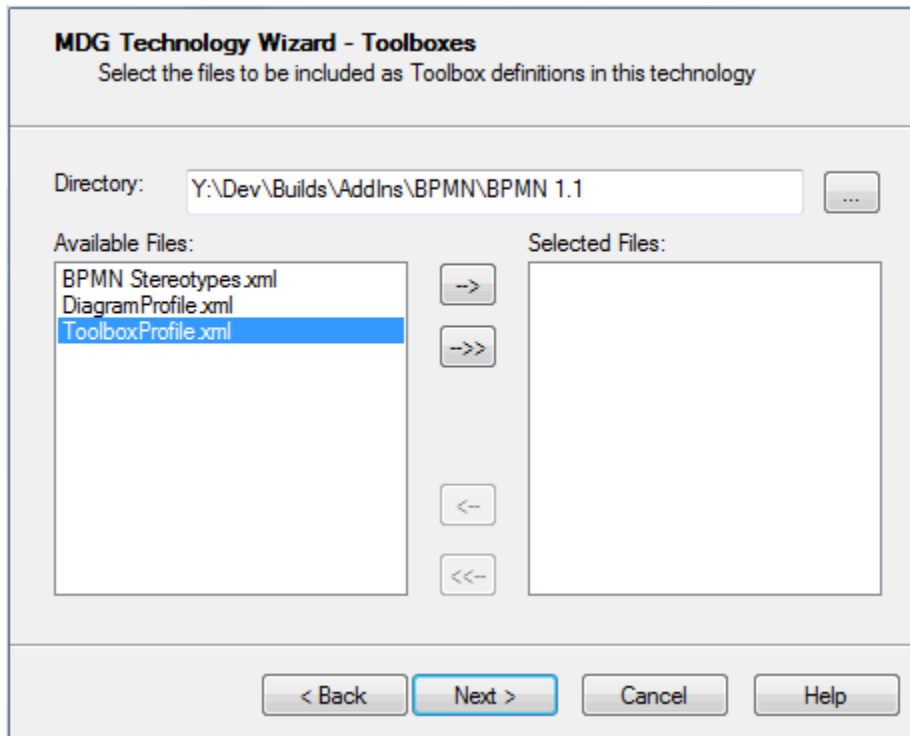


2. In the **Directory** field, navigate to the directory containing the required diagram profiles. The profiles in the directory are automatically listed in the **Available Files** panel.
3. To select each required diagram profile individually, highlight the file name in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available profiles (assuming they are all diagram profiles), click on the -->> button.
4. Click on the **Next** button to proceed.

#### 2.1.4 Add a Toolbox Profile

When creating an MDG Technology file, you can include Enterprise Architect **Toolbox** page definitions [that you have created](#)<sup>[45]</sup>. To use the Toolboxes section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Toolboxes** checkbox. The **MDG Technology Wizard - Toolboxes** dialog displays.

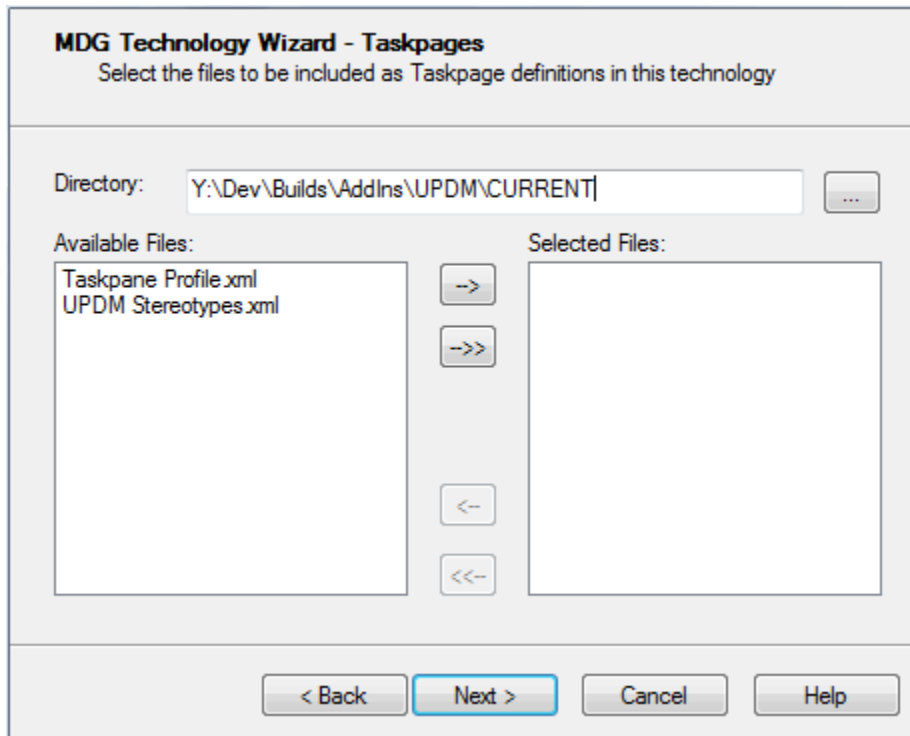


2. In the **Directory** field, navigate to the directory containing the required toolbox profiles. The profile files are automatically listed in the **Available Files** panel.
3. To select each required toolbox profile individually, highlight the file name in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available profiles (assuming they are all toolbox profiles), click on the -->> button.
4. Click on the **Next** button to proceed.

### 2.1.5 Add Task Panel Pages

When creating an MDG Technology file, you can include Enterprise Architect **Task Panel** profiles [that you have created](#)<sup>[52]</sup>. To use the Taskpages section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Taskpages** checkbox. The **MDG Technology Wizard - Taskpages** dialog displays.

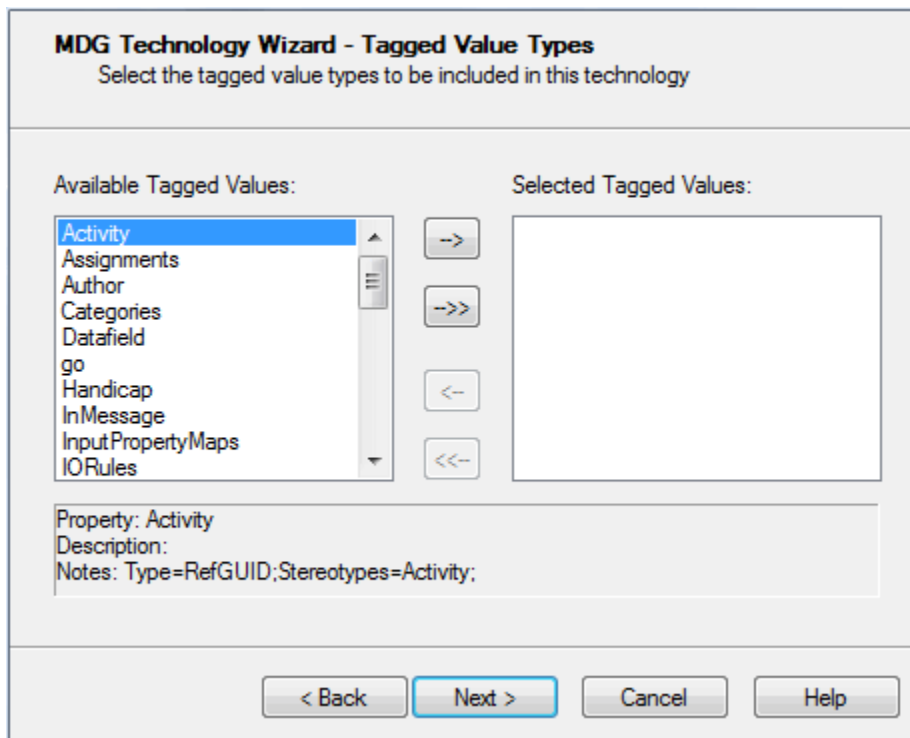


2. In the **Directory** field, navigate to the directory containing the required taskpage profiles. The profile files are automatically listed in the **Available Files** panel.
3. To select each required taskpage profile individually, highlight the file name in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available profiles (assuming they are all taskpage profiles), click on the -->> button.
4. Click on the **Next** button to proceed.

### 2.1.6 Add Tagged Value Types

When creating an MDG Technology file, you can include [Tagged Value Types](#)<sup>[79]</sup>. To use the Tagged Value Types section of the MDG Technology Types Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Tagged Value Types** checkbox. The **MDG Technology Wizard - Tagged Value Types** dialog displays.

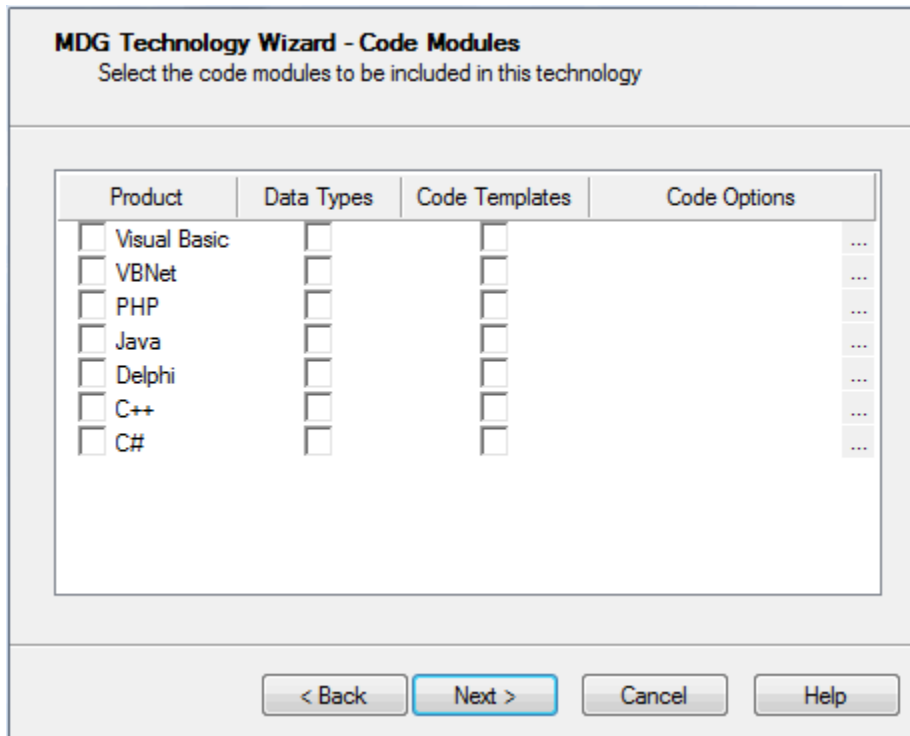


2. To select each required Tagged Value Type individually, highlight the file name in the **Available Files** list and click on the --> button. The file name displays in the **Selected Files** list. Alternatively, to select all available Tagged Value Types, click on the -->> button.
3. Click on the **Next** button to proceed.

### 2.1.7 Add Code Modules

When creating an MDG Technology file, you can include code modules. To use the code modules section of the MDG Technology Types Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Code Modules** checkbox. The **MDG Technology Wizard - Code Modules** dialog displays.



- Click on the checkboxes (**Product**, **Data Types**, and **Code Templates**) for each of the required Code Modules.

**Note:**

The code modules listed are those defined in your current project. These could be the Enterprise Architect default languages, or those you have defined yourself using code templates and the **Code Template Editor** (see *Code Engineering Using UML Models*). Before you can set up a code template for the new language in the editor, you must define at least one data type for the language (see the *Data Types* topic in *UML Model Management*). Once the MDG Technology file is created it can be loaded into your current model and into other models.

- To select any code options for a module, click on the [ ... ] button in the **Code Options** column for that module. This enables you to select an XML document that provides additional settings for the language that are not covered by the data types or code templates.

The root node of the XML document should be CodeOptions. The child nodes should be called CodeOption and should contain a *name* attribute. The supported code options are as follows:

Code Option	Description
ConstructorName	The name of a function used as a constructor. Used by the <a href="#">classHasConstructor</a> code template macro.
CopyConstructorName	The name of a function used as a copy constructor. Used by the <a href="#">classHasCopyConstructor</a> code template macro.
DefaultExtension	The default extension when generating code.
DefaultSourceDirectory	The default path to which Enterprise Architect generates new files.
DestructorName	The name of a function used as a destructor. Used by the <a href="#">classHasDestructor</a> code template macro.
Editor	The external editor used for editing source of this language.
ImplementationExtension	The extension used by Enterprise Architect to generate an

Code Option	Description
	implementation file.
ImplementationPath	The relative path from the source file to generate the implementation file.
PackagePathSeparator	The delimiter used to separate package names when using the <i>packagePath</i> macro from the code templates.

An example of a valid code options file is shown below.

```
<CodeOptions>
<CodeOption name="DefaultExtension">.ext</CodeOption>
<CodeOption name="Editor">C:\Windows\notepad.exe</CodeOption>
</CodeOptions>
```

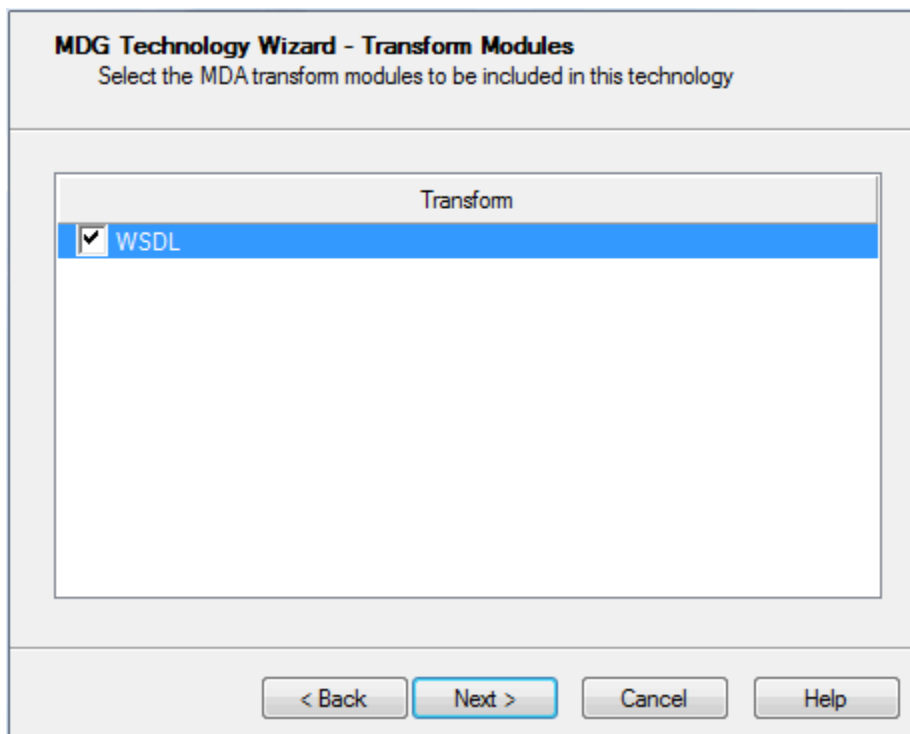
4. Click on the **Next** button to proceed.

You can edit the code option values for source code engineering and for each required language using the appropriate **Language Options** page of the **Options** dialog (see the *Code Engineering Settings* section in *Code Engineering Using UML Models*).

### 2.1.8 Add MDA Transforms

When creating an MDG Technology file, you can include the MDA Transformations that have been modified in the model. To use the Transform Modules section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#) <sup>[28]</sup> topic up to and including [Step 6](#) <sup>[31]</sup>, where you select the **MDA Transforms** checkbox. The **MDG Technology Wizard - Transform Modules** dialog displays.



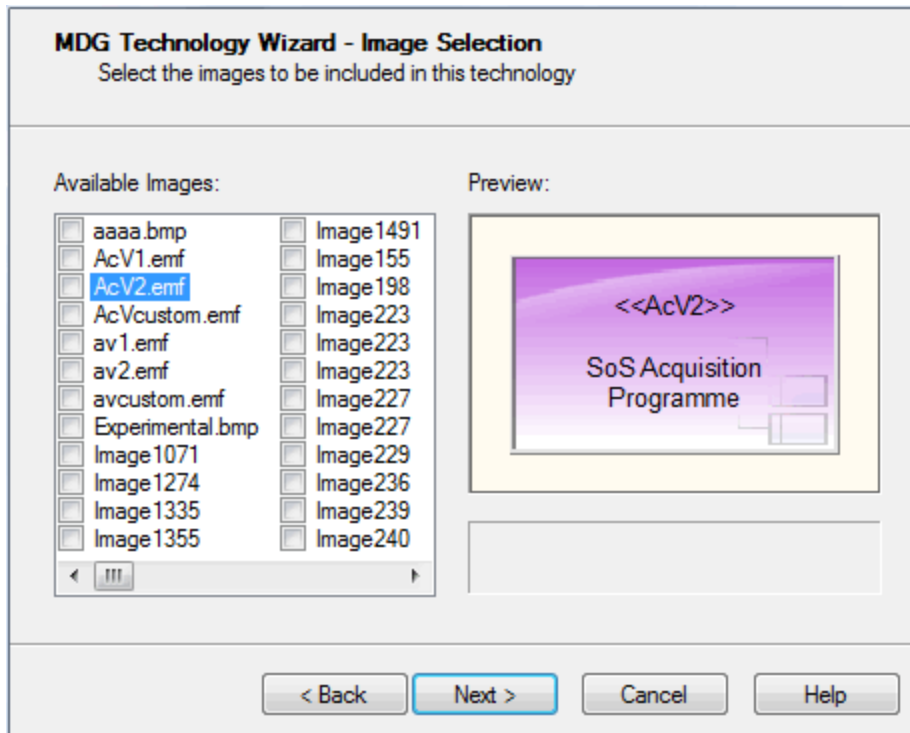
2. Click the checkbox against the template name of each required template that is present in the current model.
3. Click on the **Next** button to proceed.



### 2.1.9 Add Images

When creating an MDG Technology file, you can include the images that have been imported into the model. To use the Image Selection section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Images** checkbox. The **MDG Technology Wizard - Image Selection** dialog displays.



2. For each required model image available in the current model, select the checkbox next to the image name. A preview of each image displays on the right of the dialog as you select the checkbox.
3. Click on the **Next** button to proceed.

### 2.1.10 Add Scripts

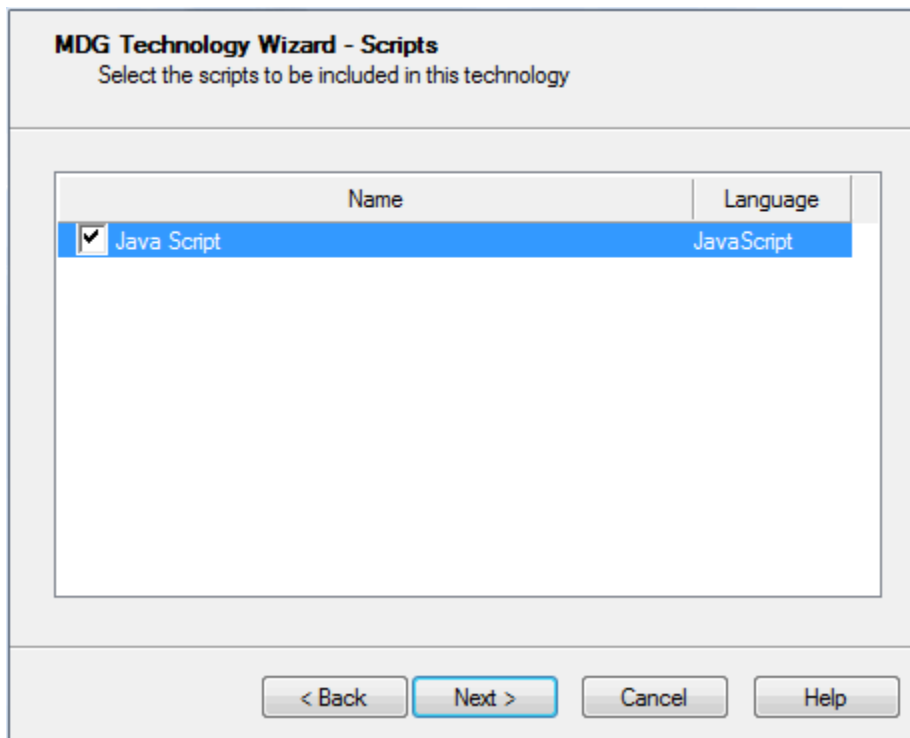
When creating an MDG Technology file, you can include scripts that you have created in the model (see *Using Enterprise Architect - UML Modeling Tool*).

**Note:**

This facility is available in the Corporate, Business and Software Engineering, Systems Engineering and Ultimate editions of Enterprise Architect.

To use the Script Selection section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Scripts** checkbox. The **MDG Technology Wizard - Scripts** dialog displays.

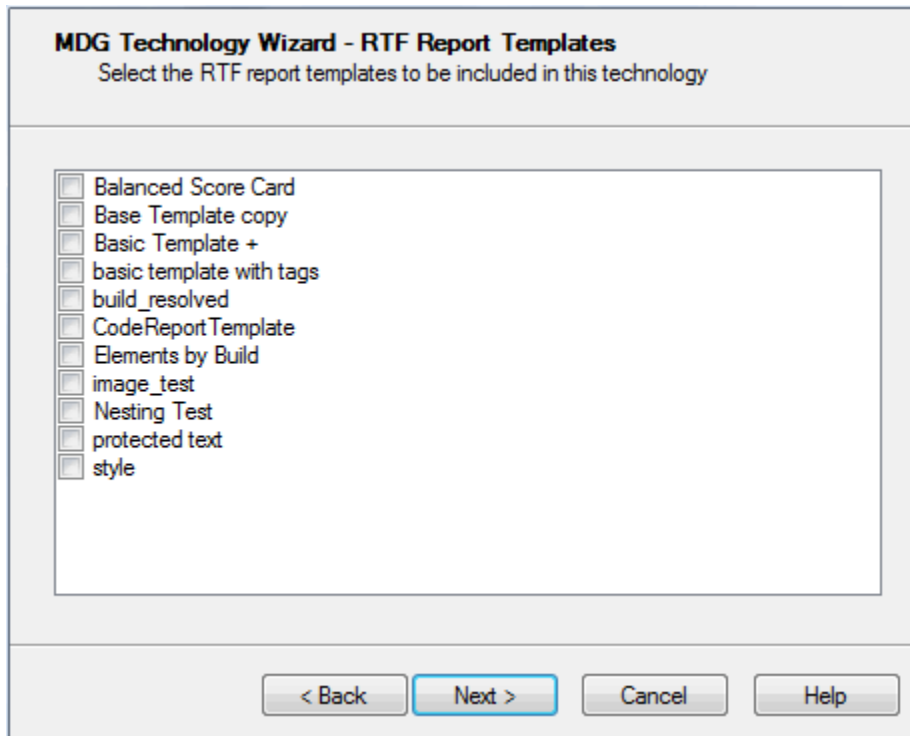


2. For each required script available in the current model, select the checkbox next to the script name.
3. Click on the **Next** button to proceed.

### 2.1.11 Add RTF Report Templates

When creating an MDG Technology file, you can include user-defined RTF Report templates. To use the report templates section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **RTF Templates** checkbox. The **MDG Technology Wizard - RTF Report Templates** dialog displays.

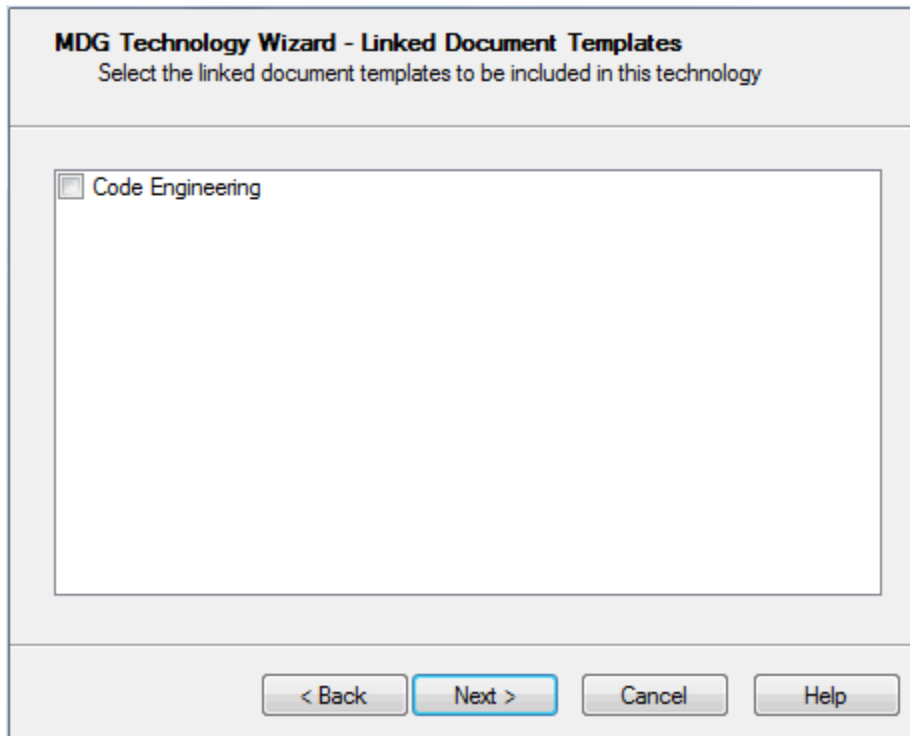


2. For each required user-defined report template available in the current model, select the checkbox next to the template name.
3. Click on the **Next** button to proceed.

### 2.1.12 Add Linked Document Templates

When creating an MDG Technology file, you can include Linked Document templates. To use the Linked Document templates section of the MDG Technology Wizard, follow the steps below:

1. Follow the steps in the [Create MDG Technologies](#)<sup>[28]</sup> topic up to and including [Step 6](#)<sup>[31]</sup>, where you select the **Linked Document Templates** checkbox. The **MDG Technology Wizard - Linked Document Templates** dialog displays.



2. For each required document template available in the current model, select the checkbox next to the template name.
3. Click on the **Next** button to proceed.

## 2.2 Working with MTS Files

An MDG Technology Selection (.MTS) file stores the selected options that you define when creating an MDG Technology File using the MDG Technology Wizard. If you use a .MTS file, you can edit it to change the features selected when you generated the file, and to add or remove the advanced features described in this topic.

### Create a .MTS File

To create a .MTS file, select the **Tools | Generate MDG Technology File** menu option to launch the MDG Technology Wizard, and work through the screens as described in [Create MDG Technologies](#)<sup>[28]</sup>. On the second page, select the **Create a new MTS file** option.

### Advanced Options For Your .MTS File

Once you have created the .MTS file, you can add:

- **Model Search** definitions (see *Using Enterprise Architect - UML Modeling Tool*)

#### Note:

If you use a custom SQL search, the SQL must include `ea_guid AS CLASSGUID` and the *object type*.

- **Model Views** (see *Using Enterprise Architect - UML Modeling Tool*)

#### Notes:

- Technology views do not store Favorite packages, only Views.
- If your exported views run searches that you have defined you must also include those searches in your MDG Technology.
- [Model Validation configurations](#)<sup>[56]</sup>
- [Model Templates](#)<sup>[57]</sup>

Open the .MTS file in a text editor. To make it easier for you, you can copy the following lines and paste them into the file before the last line of the file (that is, just before the `</MDG.Selections>` lines:

```
<ModelSearches file=""/>
<ModelViews file=""/>
```

(The code for the model validation configurations and model templates is provided in the corresponding sections, accessed via the links in the list above.)

You can, if necessary, have more than one line for each inclusion; for example, more than one *ModelSearch*. For each inserted line:

- In the file attribute, enter the filename of the Model Search XML file or Model View XML file. Save the .MTS file.

## Update the MDG Technology

Again select the **Tools | Generate MDG Technology File** menu option, but this time on the second page select **Open an Existing MTS file** and specify the file path of the .MTS file you have updated. Click on **<Next>** until the wizard is finished. Your MDG Technology file is updated.

## 2.3 Customize Toolbox Profiles

The following is a road map of how to create a set of custom toolboxes for Enterprise Architect.

1. Create a set of [Toolbox Profiles](#)<sup>[45]</sup> that contain the definitions that Enterprise Architect requires to create the toolboxes.
2. [Create a .MTS file](#)<sup>[28]</sup> containing instructions on how to build your MDG Technology. Use this .MTS file to build your MDG Technology.
3. Add some finishing touches:
  - Create [hidden sub-menus](#)<sup>[46]</sup>
  - [Override Enterprise Architect's default toolboxes](#)<sup>[47]</sup>
  - Change the default [icons for toolbox items](#)<sup>[47]</sup>.

### 2.3.1 Create Toolbox Profiles

You can create multiple toolbox profiles within an MDG Technology. Each toolbox profile contains definitions that determine what appears in the Enterprise Architect UML **Toolbox** when a specific **Toolbox** page is open, either by selecting from the **More tools...** option in the Enterprise Architect UML **Toolbox** window, or by opening or creating a diagram of the type that is linked to the toolbox profile.

To create a toolbox profile, follow the steps below:

1. Create a diagram in a profile package. Give it a name by which you can refer to it later, such as *MyClassDiagram*. In the **Notes** field for the diagram give it an alias and a description in the following format:  
Alias=MyClass;Notes=Structural elements for class diagrams;
2. On the diagram, create a Class, name it *ToolboxPage* and give it the `«metaclass»` stereotype.
3. Create a `«stereotype»` element for each of the toolbox pages to create within your toolbox, such as *MyClassElements* and *MyClassRelationships*. Set their Alias to the text to display in the title bar of each toolbox page, such as *My Class Elements* and *My Class Relationships* respectively. Use the **Notes** field to define the tool-tip for each toolbox page; that is, **Elements for Class Diagrams** and **Relationships for Class Diagrams**. Use the `«extends»` connector to set the stereotype elements to extend *ToolboxPage*. See also: [Toolbox Page Attributes](#)<sup>[46]</sup>.
4. In the `«stereotype»` elements, create an attribute for each toolbox item. The name of the attribute should be the name of the element or connector to be dropped, including namespace, for example, *UML::Package*, *UML::Class* and *UML::Interface*. The toolbox items display in the same order as the attributes in the Class, so make use of the attribute ordering buttons to define the order of your toolbox.

**Note:**

To name an attribute for an item from your own technology, precede it with your profile name as the namespace, and then follow it in brackets with the element or connector type that you are extending (so that Enterprise Architect knows what object to create). For example, a SysML block element would appear as *SysML::Block(UML::Class)*. Click on the following links for a complete list of [elements](#)<sup>[48]</sup> and [connectors](#)<sup>[49]</sup> that can be extended.

To define a toolbox item that allows a pattern to be dropped onto a diagram, name the attribute *MyTechnology::MyPattern(UMLPattern)* where *MyTechnology* is the name of the technology and *MyPattern* is the name of the pattern to drop. For example, *GoFPatterns::Builder(UMLPattern)*.

You might not want to use names such as *UML::Package* or *UML::Class* in your toolbox, so give the attributes an **Initial Value** of, for example, *Package* or *Class*.

5. To save the toolbox profile, right-click on the diagram and select the **Save as Profile** context menu option.

**Note:**

Each profile element incorporated into an MDG **Toolbox** page enables synchronization of the Tagged Values and Constraints of all elements created from them (see *Extending UML With Enterprise Architect*).

### 2.3.1.1 Toolbox Page Attributes

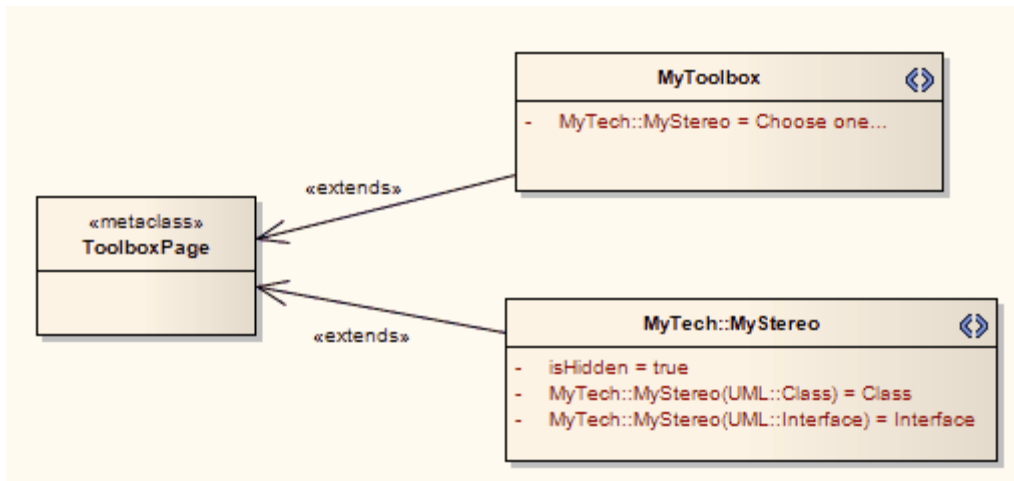
The following attributes can be added to a stereotype Class that extends the *ToolboxPage* metaclass:

- **imagesOnly**: if you give a toolbox page an attribute named *imagesOnly* with **Initial Value** set to **true**, the toolbox page displays without the text labels next to the icons
- **isCommon**: if you give a toolbox page an attribute named *IsCommon* with **Initial Value** set to **true**, the toolbox page is common to all defined toolboxes while your technology is active
- **isCollapsed**: if you give a toolbox page an attribute named *IsCollapsed* with **Initial Value** set to **true**, the toolbox page is initially minimized.
- **Icon**: see [Icons for Toolbox Items](#)<sup>[47]</sup>
- **isHidden**: see [Create Hidden Sub-Menus](#)<sup>[46]</sup>.

### 2.3.2 Create Hidden Sub-Menus

To create a sub-menu, create an additional «*stereotype*» element in the same toolbox profile and give it an attribute named *isHidden* with **Initial Value** of **true**. Define the toolbox item attributes as before. In the parent «*stereotype*» element, create an attribute with the identical name to the sub-menu element. The sub-menu element can have an alias.

This technique is very useful for 'disambiguating' stereotypes that can be applied to multiple metaclasses. In the example below, the «*MyStereo*» stereotype can be applied to either a Class or an Interface. On dragging and dropping one from the toolbox, a hidden menu displays giving the choice of Class or Interface, then the appropriate element is dropped:



### 2.3.3 Override Default Toolboxes

Enterprise Architect has many default Toolbox Profiles, one for each of its inbuilt diagram types. These define the Toolbox pages that are displayed, by default, every time a diagram of a specific type is opened or brought into view.

To replace one of Enterprise Architect's default toolboxes with one of your own (for example, if you have your own version of the `UML::Class` toolbox that you want to be opened every time a Class diagram is opened - as long as your technology is active) then include a *RedefinedToolbox* clause in the **Notes** field for the diagram properties of your Toolbox Profile diagram. For example, the profile diagram's **Notes** field could resemble the following:

```
RedefinedToolbox=UML::Class;Alias=Class;Notes=Structural elements for Class diagrams;
```

This states that the profile is the new Enterprise Architect default for all UML Class diagrams. For a list of inbuilt diagram types, see [the names of the toolboxes](#)<sup>[48]</sup>.

### 2.3.4 Assign Icons To Toolbox Items

To assign an icon to a toolbox item, create a new `«stereotype»` element in the same toolbox profile as the toolbox item. Have the stereotype element extend a `«metaclass»` element named `ToolboxItemImage`. The `«stereotype»` element must have the same name as the attribute that it is assigning an image to (for example, `MyTech::MyStereo(UML::Class)` in the diagram below) and must have an attribute named `Icon` with **Initial Value** set to the full path and file name of the image to be used. The image must be a 16x16 .BMP file.



### 2.3.5 Enterprise Architect Toolboxes

The following is a list of the Enterprise Architect UML Toolboxes that can be overridden:

- UML::Activity
- UML::Class
- UML::Communication
- UML::Component
- UML::Composite
- UML::Deployment
- UML::Interaction
- UML::Metamodel
- UML::Object
- UML::Profile
- UML::State
- UML::Timing
- UML::UseCase
- Extended::Analysis
- Extended::Custom
- Extended::DataModeling
- Extended::Maintenance
- Extended::Requirements
- Extended::UserInterface
- Extended::WSDL
- Extended::XMLSchema

### 2.3.6 Elements Used in Toolboxes

The following elements (all preceded with the namespace **UML::**) can be extended or redefined in Enterprise Architect **Toolbox** pages. The text in red indicates the label name displayed in the default Enterprise Architect **Toolbox** pages, where this differs in any way from the **UML::** statement text.

When these profile elements are incorporated into an MDG **Toolbox** page, they enable synchronization of the Tagged Values and Constraints of all elements created from them (see *Extending UML With Enterprise Architect*).

You can also extend [connectors](#) <sup>49</sup>.

- Action
- Activity
- ActivityFinal (**Final**)
- ActivityInitial (**Initial**)
- ActivityParameter
- ActivityPartition (**Partition**)
- ActivityRegion (**Region**)
- Actor
- Artifact
- AssociationElement (**Association**)
- Boundary (for Use Cases)
- CentralBufferNode (**Central Buffer Node**)
- Change
- Choice
- Class
- InteractionState (**State/Continuation**)
- Interface
- Issue
- Junction
- Lifeline
- MergeNode (**Merge**)
- MessageEndPoint (**Endpoint** or **Message Endpoint**)
- MessageLabel (**Message Label**)
- Metaclass
- Node
- Object
- ObjectBoundary (**Boundary**)
- ObjectControl (**Control**)
- ObjectEntity (**Entity**)
- Package



- Collaboration
- CollaborationOccurrence
- Comment (**Note**)
- Component
- Constraint
- Datastore
- Decision
- DeploymentSpecification (**Deployment Specification**)
- Device
- DiagramLegend (**Diagram Legend**)
- DiagramNotes (**Diagram Notes**)
- DocumentArtifact (**Document Artifact** or **Document**)
- Entity (**Information**)
- EntityObject (**Entity**)
- EntryState (**Entry**)
- Enumeration
- ExceptionHandler (**Exception**)
- ExecutionEnvironment (**Execution Environment**)
- ExitState (**Exit**)
- Feature
- FinalState (**Final**)
- FlowFinalNode (**Flow Final**)
- ForkJoinH (**Fork/Join** - Horizontal)
- ForkJoinV (**Fork/Join** - Vertical)
- Gate (**Diagram Gate**)
- GUIElement (**UI Control**)
- HistoryState (**History**)
- Hyperlink
- InformationItem (**Information Item**)
- InitialState (**Initial**)
- InteractionFragment (**Fragment**)
- PackagingComponent
- Part
- Port
- Primitive
- Process
- Profile
- ProvidedInterface (**Expose Interface**)
- ReceiveEvent (**Receive**)
- Requirement
- RobustBoundary (**Boundary**)
- RobustControl (**Control**)
- RobustEntity (**Entity**)
- Screen
- SendEvent (**Send**)
- SequenceBoundary (**Boundary**)
- SequenceControl (**Control**)
- SequenceEntity (**Entity**)
- Signal
- State
- StateMachine (**State Machine**)
- StateTimeLine (**State Lifeline**)
- Stereotype
- StructuredActivity (**Structured Activity**)
- SynchState (**Synch**)
- Table
- Terminate
- TestCase (**Test Case**)
- Text
- UseCase (**Use Case**)
- UMLBoundary (**Boundary**)
- ValueTimeLine (**Value Lifeline**)

### 2.3.7 Connectors Used In Toolboxes

The following connectors (all preceded with the namespace **UML::**) can be extended or redefined in Enterprise Architect toolboxes. The text in red indicates the label name displayed in the default Enterprise Architect **Toolbox** pages, where this differs in any way from the **UML::** statement text.

You can also extend [elements](#)<sup>48</sup>.

- Aggregation (**Aggregate**)
- Assembly
- NoteLink (**Note Link**)
- ObjectFlow (**Object Flow**)

- Association (**Associate**)
- AssociationClass (**Association Class**)
- CallFromRecursion (**Call**)
- CommunicationPath (**Communication Path**)
- Composition (**Compose**)
- Connector
- ControlFlow (**Control Flow**)
- Delegate
- Dependency
- Deployment
- Extension
- Generalization (**Generalize** or **Inheritance**)
- InformationFlow (**Information Flow**)
- InterruptFlow (**Interrupt Flow**)
- Invokes
- Manifest
- Message
- Nesting
- Occurrence
- PackageImport (**Package Import**)
- PackageMerge (**Package Merge**)
- Precedes
- ProfileApplication (**Application**)
- Realization (**Realize** or **Implements**)
- Recursion
- Redefinition
- Representation
- Represents
- RoleBinding (**Role Binding**)
- SelfMessage (**Self-Message**)
- TagValAssociation (**Tagged Value**)
- TraceLink (**Trace**)
- Transition
- UCExtend (**Extend**)
- UCInclude (**Include**)
- UseCaseLink (**Use**)

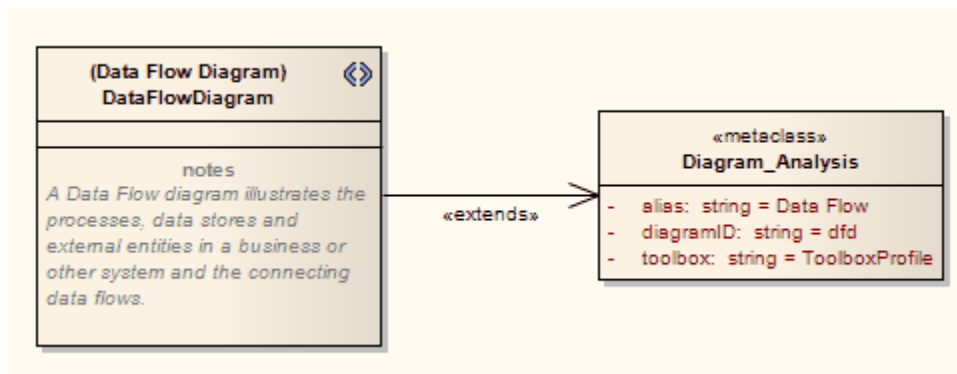
## 2.4 Create Diagram Profiles

### Custom Diagram Types

You can create extended diagram types in Enterprise Architect and include them in MDG Technologies. To do this, perform the following steps.

1. Create a profile with the same name as the MDG Technology in which it is to be included; for example, *SysML*.
2. Create a «*stereotype*» Class element that is named as the custom diagram, for example, *BlockDefinition*.
3. Create a Class element and name it as one of the [Built-In Diagram Types](#)<sup>[51]</sup> prefixed with *Diagram\_*, for example *Diagram\_Logical* for Class diagrams or *Diagram\_Use Case* for Use Case diagrams.
4. Give the *Diagram\_x* Class the «*metaclass*» stereotype and draw an «*extends*» connector from the stereotype to the metaclass.
5. In the **Notes** field, give the stereotype Class a brief description of what the diagram is used for. This description displays in the bottom right-hand corner of the **New Diagram** dialog.
6. Give the *Diagram\_x* Class the following attributes as required:
  - *alias*: *string = Type* (where *Type* appears before the word 'Diagram' on the diagram title bar)
  - *diagramID*: *string = abc* (where *abc* is the diagram type that appears in the diagram frame label - see *The UML Dictionary*)
  - *toolbox*: *string = ToolboxName* (where *ToolboxName* is the name of the toolbox profile for the toolbox that opens automatically each time a diagram is opened)
  - *frameString*: *string = FrameFormatString* (where *FrameFormatString* is a string containing substitution macros for defining the frame title, with or without additional delimiters such as [ ]; m across that can be used are:
    - #DGMSTEREO#
    - #DGMID#
    - #DGMTYPE#
    - #DGMALIAS#

- #DGMOWNERNAME#
  - #DGMOWNERNAMEFULL#
  - #DGMNAME#
  - #DGMNAMEFULL#
  - *swimlanes*: *string* = *Lanes=2;Orientation=Horizontal;Lane1=Title1;Lane2=Title2*; (where *Lanes* can be any value, but the number of *Lane<n>* values must equal the value of *Lanes*; *Orientation* can be omitted, in which case the swimlanes default to vertical)
  - *styleex*: *string* = one or more of a range of values; see [Attribute Values - stylex & pdata](#)<sup>[51]</sup>
  - *pdata*: *string* = one or more of a range of values; see [Attribute Values - stylex & pdata](#)<sup>[51]</sup>
- The following example shows the DFD diagram profile which defines a DFD diagram as an extension of the Enterprise Architect Analysis diagram.



7. Save the diagram as a profile in the usual manner.
8. [Add the diagram profile to the](#)<sup>[34]</sup>.MTS file used in the MDG Technology.

### 2.4.1 Built-In Diagram Types

The following is a full list of built-in diagram types provided by Enterprise Architect.

- Activity
- Analysis
- Collaboration
- Component
- CompositeStructure
- Custom
- Deployment
- InteractionOverview
- Logical
- Object
- Package
- Sequence
- Statechart
- Timing
- Use Case

Note the use of *Logical* for Class diagrams and also notice the space in the middle of *Use Case*. These names are used in [Defining Child Diagram Types](#)<sup>[21]</sup>, or prefixed by *Diagram\_* in creating [Diagram Profiles](#)<sup>[50]</sup>.

### 2.4.2 Attribute Values - stylex & pdata

When creating a diagram profile, you can use the *pdata* and *stylex* attributes to define a range of characteristics of the diagrams created with the profile. If the attribute is defining several characteristics at once, put the values in a single string separated by semicolons. For example:

*styleEx: string = HideQuals=0;AdvanceElementProps=1;ShowNotes=1;*

**styleex: string =**

- *TConnectorNotation=Option;* (where *Option* is one of **UML 2.1**, **IDEF1X**, or **Information Engineering**)
- *ShowAsList=1;* (to make the diagram open directly into the **Element List** - see the *View Options* topic in *Using Enterprise Architect - UML Modeling Tool*)
- *AdvanceElementProps=1;* (to show the element property string)
- *ShowTests=1;* (to show the element Testing compartment)
- *ShowMaint=1;* (to show the element Maintenance compartment)
- *ShowNotes=1;* (to show the element Notes compartment)
- *HideQuals=0;* (to show qualifiers and visibility indicators)
- *AdvancedFeatureProps=1;* (to show the feature property string)
- *ShowOpRetType=1;* (to show the operation return type)
- *SuppressBrackets=1;* (to suppress brackets on operations without parameters)
- *AttPkg=1;* (to show package visible class members)
- *VisibleAttributeDetail=1;* (to show attribute details on the diagram)
- *TExplicitNavigability=1;* (to show non-navigable connector ends)
- *AdvancedConnectorProps=1;* (to show connector property strings)
- *SuppConnectorLabels=1;* (to suppress all connector labels).

**pdata: string =**

- *UseAlias=1;* (to use the aliases or elements in the diagram, if available)
- *HideParents=0;* (to show additional parents of elements in the diagram)
- *HideEStereo=0;* (to show element stereotypes in the diagram)
- *ShowSN=1;* (to show sequence notes)
- *ShowIcons=1;* (to use stereotype icons)
- *HideAtts=0;* (to show the element Attributes compartment)
- *HideOps=0;* (to show the element Operations compartment)
- *ShowTags=1;* (to show the element Tagged values compartment)
- *ShowReqs=1;* (to show the element Requirements compartment)
- *ShowCons=1;* (to show the element Constraints compartment)
- *HideStereo=0;* (to show attribute and operation stereotypes)
- *HideProps=0;* (to show property methods)
- *OpParams=3;* (to show operation parameters)
- *HideRel=0;* (to show relationships)
- *SuppCN=0;*(to show collaboration numbers).

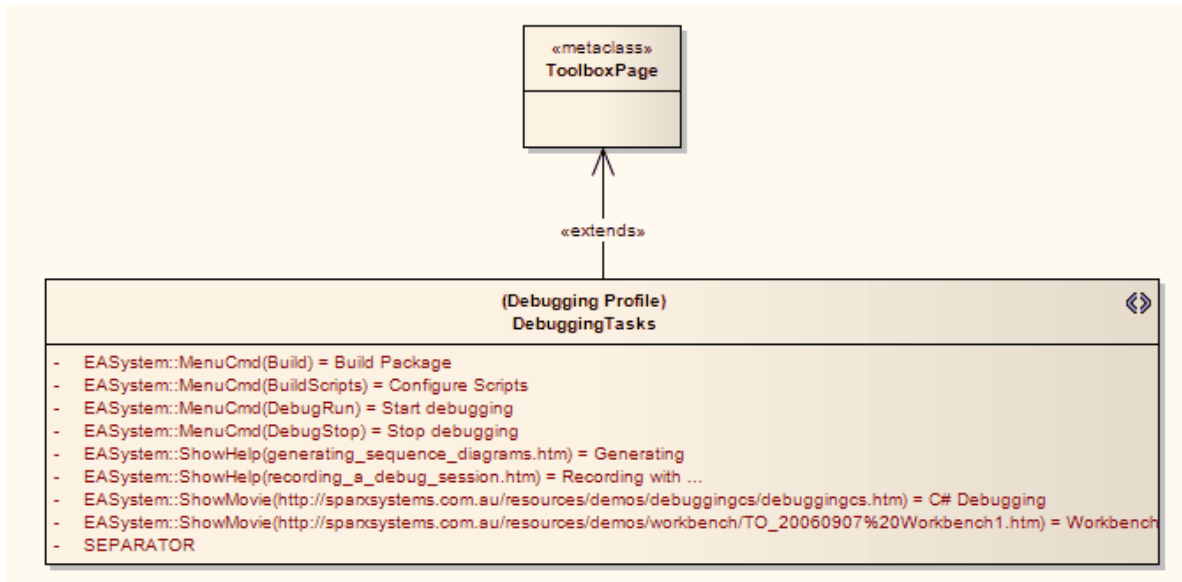
## 2.5 Create Tasks Pane Profiles

Defining **Tasks Pane** profiles is a four-part process:

1. [Define Toolboxes](#)<sup>[53]</sup>. Create any number of stereotype elements that each define a **Tasks Pane** toolbox page
2. [Define Contexts](#)<sup>[55]</sup>. Create any number of stereotype elements that each define a named Context. A context might be when a specific diagram type is open, or when a specific element type is selected.
3. [Allocate Contexts](#)<sup>[55]</sup> to Toolboxes. Define the many-to-many relationships between the **Tasks Pane** toolboxes and the available contexts.
4. [Create the Profile](#)<sup>[56]</sup> and [incorporate it](#)<sup>[36]</sup> into your Technology.

## 2.5.1 Define Tasks Pane Toolboxes

A **Tasks Pane** toolbox is defined by a «stereotype» Class that extends a «metaclass»ToolboxPage element. These elements must be owned by a «profile» package. Each «stereotype» Class represents the contents of the **Tasks Pane** for a given context, and each attribute of the «stereotype» Class defines a command button in the **Tasks Pane**. The following diagram shows an example of a **Tasks Pane** toolbox.



The title bar of the **Tasks Pane** toolbox is defined by the Alias of the «stereotype» Class, in this case *Debugging Profile*. This example uses the following standard attribute types:

- **EASystem::MenuCmd**. These entries name an Enterprise Architect main menu command inside round brackets. See the complete [list of inbuilt commands](#)<sup>[53]</sup>. Type the text to appear in the **Tasks Pane** into the **Initial Value** field.
- **EASystem::ShowHelp**. These entries name a page from the *Enterprise Architect User Guide* inside round brackets. To find out the names of pages in the *Enterprise Architect User Guide*, right-click on the page and select the **Properties** context menu option. Type the text to appear in the **Tasks Pane** into the **Initial Value** field.
- **EASystem::ShowMovie**. These entries give the URL of a movie inside round brackets. Type the text to appear in the **Tasks Pane** into the **Initial Value** field.
- **SEPARATOR**. This entry indicates that a separator should be placed in the **Tasks Pane** toolbox. If it is necessary to place multiple separators in a single toolbox, note that Enterprise Architect does not allow identically named attributes for a Class: simply change the case of one or more letters to get around the problem.

Other useful attributes include:

- **EASystem::ShowURL**. This entry gives the URL of a web page inside round brackets. Type the text to appear in the **Tasks Pane** into the **Initial Value** field.
- **isCommon**: A boolean attribute with **Initial Value** set to **True**, defines a **Tasks Pane** toolbox as context-free and common, appearing for all contexts
- You can also [run Add-In functions](#)<sup>[54]</sup> from the **Tasks Pane**.

### Next Step

The next step is to create a set of [Tasks Pane Contexts](#)<sup>[55]</sup>.

#### 2.5.1.1 Built-In Tasks Pane Commands

The following Enterprise Architect commands can all be used in user-defined **Tasks Pane** profiles. **Tasks Pane** pages have attributes named in the form `EASystem::MenuCmd(<CommandName>)` where `<CommandName>` is the name chosen from the following list:

- AddDiagram
- AddElement
- AddPackage
- AutoRecordThread
- AddModelFromPattern
- Build
- BuildScripts
- ConfigureCSV
- ConfigureValidation
- CreateBaseLine
- CreateSequenceDiagram
- DebugPause
- DebugRun
- DebugStop
- Deploy
- DiagramsOnlyReport
- ElementUsage
- ExportXML
- FileNew
- FileOpen
- GenerateDDL
- GenerateWSDL
- GenerateXMLSchema
- ImplementationDetails
- ImportBinary
- ImportExportCSV
- ImportSchema
- ImportSourceDirectory
- ImportWSDL
- ImportXML
- ImportXMLSchema
- Run
- RunHTMLReport
- RunRTFReport
- SetClassifier
- ShowHideExecution
- StartDebugRecording
- StepInto
- StepOut
- StepOver
- StopDebugRecording
- Test
- TestingReport
- ToggleLevelNumbering
- TransformPackage
- TransformSelectedElements
- ValidateModel
- ViewAuditing
- ViewDebug
- ViewElementList
- ViewForum
- ViewHierarchy
- ViewMaintenance
- ViewOutput
- ViewProjectManagement
- ViewRelationships
- ViewRelMatrix
- ViewRequirementTypes
- ViewRules
- ViewSearch
- ViewSourceCode
- ViewTaggedValues
- ViewTesting
- ViewTestingDetails
- ViewWebBrowser

### 2.5.1.2 Run Add-In Functions

To run Add-In functions from the **Tasks Pane**, you create an attribute in the **Tasks Pane «stereotype» Class** with the following format:

```
"Assembly::FunctionName()"
```

where *Assembly* is the name of the Add-In and *FunctionName* is the name of a public function in the Add-In. Give the attribute an initial value of the text that is to appear in the **Tasks Pane**. The function receives two parameters and returns a success status, as in the following VB.Net example:

```
Public Function ShowMyDiagram(ByRef Repository As EA.Repository, ByVal args As Object) As String
    Dim ret As String
    ret = Repository.SQLQuery("select ea_guid from t_diagram where diagram_type='Custom' and StyleEx like
    *;MDGDgm=MyDiagrams::MyCustomDiagram;*")
    If ret Is Nothing Then
        ShowMyDiagram = False
        Exit Function
    End If

    Dim oXML As MSXML2.DOMDocument = New MSXML2.DOMDocument
    oXML.loadXML(ret)

    Dim NodeList As MSXML2.IXMLDOMNodeList = oXML.selectNodes("//ea_guid")
    If NodeList.length = 0 Then
        ShowMyDiagram = False
        Exit Function
    End If
```

```

Dim Node As MSXML2.IXMLDOMNode
Dim diag As EA.Diagram
If NodeList.length >= 1 Then
    Node = NodeList.item(0)
    diag = Repository.GetDiagramByGuid(Node.text)
    Repository.OpenDiagram(diag.DiagramID)
    Repository.ShowInProjectView(diag)
End If

ShowMyDiagram = True
End Function

```

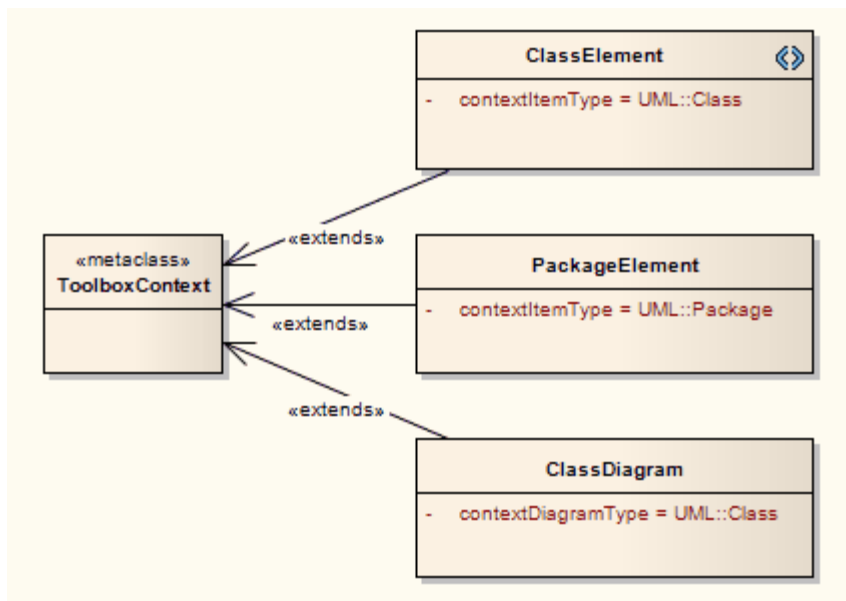
## 2.5.2 Define Tasks Pane Contexts

### Named Contexts

To create a context-sensitive set of **Tasks Panes**, you must define a set of named contexts that can be used in the definition. A named context is a *«stereotype» Class* which extends a *«metaclass»* named *ToolboxContext*. These elements must be owned by the same *«profile»* package that owns the [Task Pane toolbox definitions](#) [53]. The context Class has one of the following attributes:

- contextDiagramType; this should have an **Initial Value** set to a valid diagram type
- contextItemType; this should have an **Initial Value** set to a valid element type
- contextKey.

#### Example Context Profile

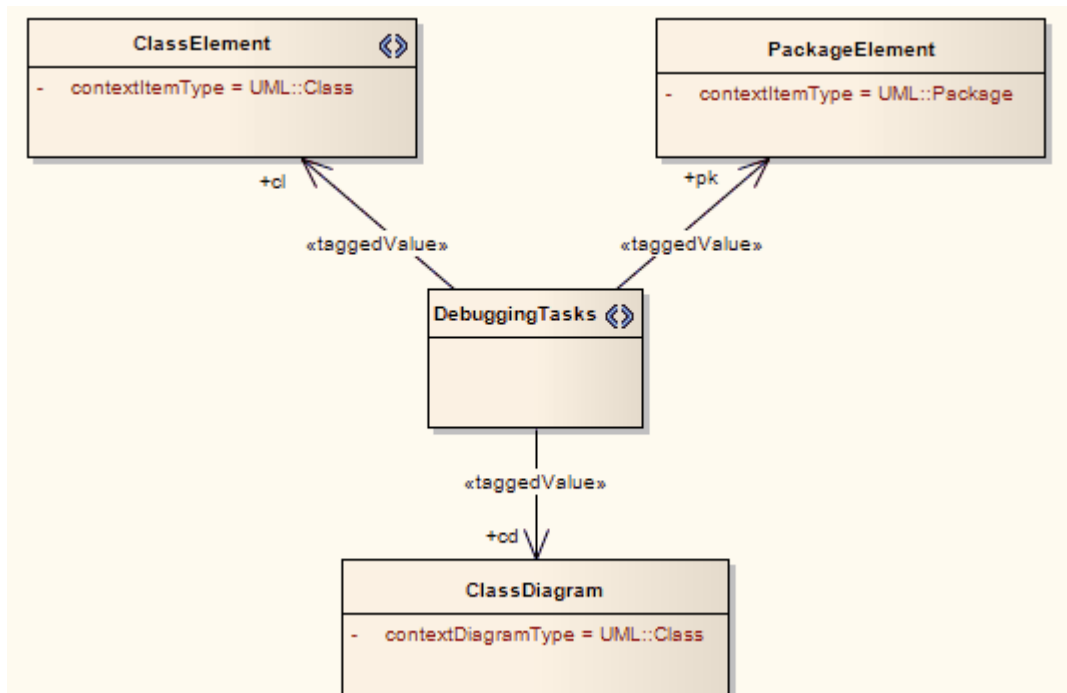


### Next Step

The next step is to [allocate Tasks Pane contexts to Tasks Pane toolboxes](#) [55].

## 2.5.3 Allocate Tasks Pane Contexts

Once you have defined your [Tasks Pane toolboxes](#) [53] and [Tasks Pane contexts](#) [55], you can allocate your toolboxes to as many contexts as apply, and any number of toolboxes can be allocated to a single context. This is done by creating a *«taggedValue»* connector from the toolbox element to the context element. The association end at the context end must be named. The following diagram shows how this might appear.



### Next Step

The next step is to [Save your Tasks Pane Profile](#) <sup>56</sup>.

## 2.5.4 Save a Tasks Pane Profile

The best organization structure for the model in which you are creating your **Tasks Pane** Profile is:

- A single «profile» package
- Three diagrams within the «profile» package named *Toolboxes*, *Contexts* and *Context Allocations*
- Each toolbox page «stereotype» element is owned by the «profile» package and appears on the *Toolboxes* and *Context Allocations* diagrams
- Each context «stereotype» element is owned by the «profile» package and appears on the *Contexts* and *Context Allocations* diagrams
- Each «metaclass» element is owned by the «profile» package and appears on the *Toolboxes* or *Contexts* diagram.

From this structure, creating a **Tasks Pane** Profile is as simple as right-clicking on the «profile» package in the **Project Browser** and selecting the **Save Package as UML Profile** context menu option.

## 2.6 Define Validation Configuration

The **Model Validation Configuration** dialog can be opened using the **Project | Model Validation | Configure...** menu option. Using this dialog, you can choose which sets of validation rules are and are not executed when a user performs a validation. Rather than perform this configuration manually and potentially have to change the settings every time Enterprise Architect is started and a different technology is set active, you can define the configuration settings within the MTS file.

To specify a set of rules as a white-list (that is, anything added to this list is turned ON), open your MTS file in a text editor and copy and paste the following <ModelValidation> block at the top level inside the <MDG.Selections> block:

```

<ModelValidation>
  <RuleSet name="BPMNRules"/> <!-- ruleset ID defined in the Project.DefineRuleCategory call -->
  <RuleSet name="MVR7F0001"/> <!-- notice you can turn on/off system rules as well! -->
</ModelValidation>
  
```



Ensure that the ruleset IDs do not contain any spaces.

To specify a set of rules as a black-list (that is, anything added to this list is turned OFF), open your MTS file in a text editor and copy and paste the following <ModelValidation> block at the top level inside the <MDG.Selections> block:

```
<ModelValidation isBlackList="true">
  <RuleSet name="BPMNRules"/>
  <RuleSet name="MVR7F0001"/>
</ModelValidation>
```

In the examples above, "BPMNRules" is the rule-set ID defined in the Project.DefineRuleCategory call - see [Project Interface](#)<sup>[262]</sup> for details. "MVR7F0001" is one of Enterprise Architect's built-in rule-sets. These validation options are applied when you activate the appropriate technology. The global (default) technology has all rules turned on.

## 2.7 Incorporate Model Templates

Enterprise Architect has a number of Model Templates (see *Using Enterprise Architect - UML Modeling Tool*) that can be added into a model, either on creation of the model, or at any time by right-clicking on a package in the **Project Browser** and selecting the **Add | Add a New Model using Wizard...** context menu option (see *UML Model Management*). You can create your own templates and include them in your MDG Technology. The first step is to create a template package and save it to the MTS file.

Open your MTS file in a text editor and copy and paste the following <ModelTemplates> block at the top level inside the <MDG.Selections> block:

```
<ModelTemplates>
  <Model name="Template Name"
    description="This is the description."
    location="MyTemplatePackage.xml"
    default="yes"
    icon = "34"
    filter= "Filter Name"/>
</ModelTemplates>
```

You can include as many <ModelTemplates> blocks as you have model templates. The attributes have the following meanings:

- **Model name:** The name of the model template as shown in the **Select model(s)** dialog, which displays when you create a new model or when you execute the **Add a New Model using Wizard** menu option.
- **description:** The text that is displayed in the **Select model(s)** dialog when the name is selected.
- **location:** Contains the path of the XML file that contains the XMI export of the model template package, relative to the location of the MDG Technology file. If the XMI file is in the same folder as the technology file then this just contains the file name.
- **default:** Contains either **yes** indicating that the model template is checked by default, or **no** indicating that the model template is un-checked by default.
- **icon:** Contains an index to Enterprise Architect's base icons list. To show the appropriate view icon, use one of the following values: **29** = Use Case, **30** = Dynamic; **31** = Class; **32** = Component; **33** = Deployment; **34** = Simple.
- **filter:** If you have a large number of model templates, you can group them on the **Select model(s)** dialog by giving all the model templates in the same group the same filter name. The filter name given appears in the **Select from:** list box in the **Select model(s)** dialog.

## 2.8 Deploy An MDG Technology

An MDG Technology can be deployed in one of two ways: as a file or from an Add-In.

### Deploy From a File

To deploy your technology as a file, you have a number of choices:

- Copy it to a folder named MDGTechnologies, which you must create under your Enterprise Architect installation directory (by default this is C:\Program Files\Sparx Systems\EA. When you restart Enterprise Architect, your MDG Technology is deployed.
- Copy it to any folder in your file system, including network drives: use the Enterprise Architect **Settings | MDG Technologies...** menu option, press the **Advanced** button and add the folder to the Technologies path. This deployment method enables you to quickly and easily deploy a technology to all Enterprise

Architect users on a LAN.

- Upload it to an internet or intranet location: use the Enterprise Architect **Settings | MDG Technologies...** menu option, press the **Advanced** button and add the URL to the Technologies path. This deployment method enables you to quickly and easily deploy a technology to an even wider group of Enterprise Architect users.

### Deploy From an Add-in

To deploy your technology from an Add-In, you must write an [EA\\_OnInitializeTechnologies](#)<sup>[147]</sup> function. The following example is written in VB.Net:

```
Public Function EA_OnInitializeTechnologies(ByVal Repository As EA.Repository) As Object
    EA_OnInitializeTechnologies = My.Resources.MyTechnology
End Function
```

## 3 Shape Scripts



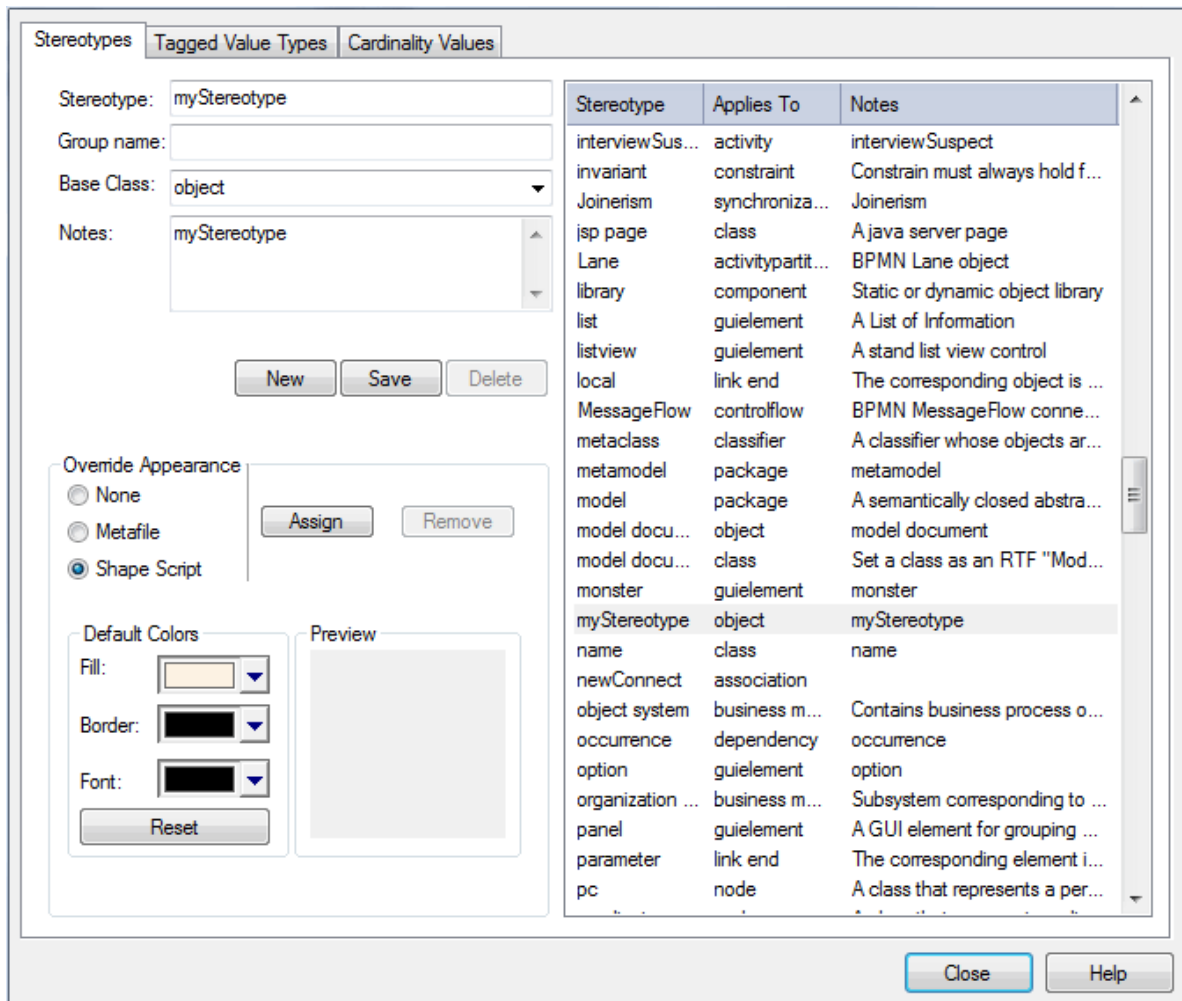
### Introduction

Enterprise Architect *Shape Scripts* enable you to specify custom shapes via a scripting language. These custom shapes are drawn instead of the standard UML notation. Each script is associated with a particular stereotype, and is drawn for every element of that stereotype. The following topics describe how to create and apply Shape Scripts:

- [Getting Started with Shape Scripts](#) <sup>59</sup>
- [Write Scripts](#) <sup>63</sup>
- [Example Scripts](#) <sup>75</sup>
- [Shape Editor](#) <sup>62</sup>
- [Add Shape Scripts to UML Profiles](#) <sup>14</sup>

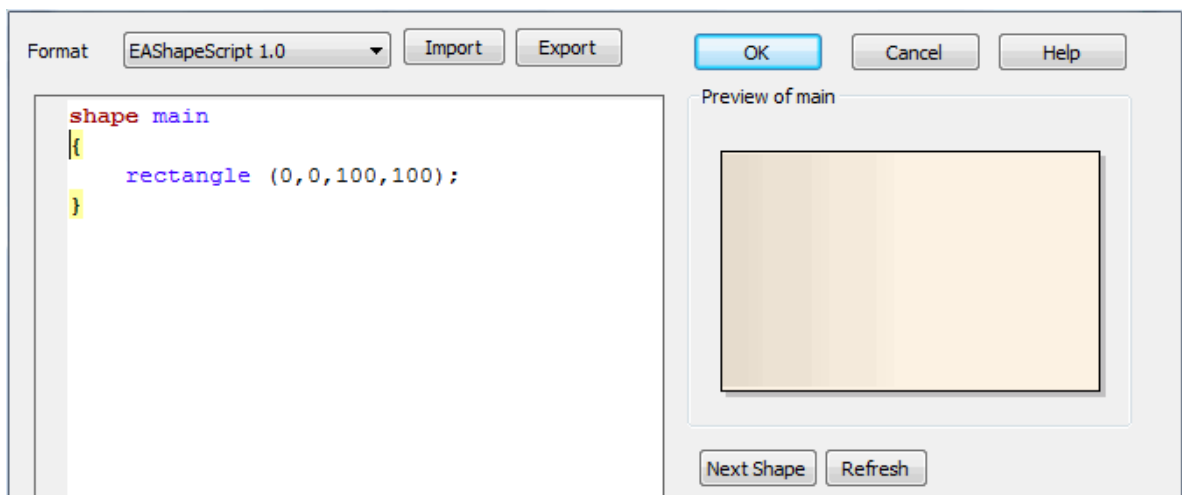
### 3.1 Getting Started With Shape Scripts

[Shape Scripts](#) <sup>59</sup> are associated with stereotypes and are defined via the **Stereotypes** tab of the **UML Types** dialog. To access this dialog, select the **Settings | UML** menu option. Each stereotype defined can have a Shape Script.



You can create a Shape Script for an existing stereotype by selecting the stereotype from the list. Alternatively, you can create new stereotypes by clicking on the **New** button and giving the stereotype a name. Select a base Class and click on the **Save** button. Once the stereotype is saved, it displays in the list.

To override the appearance, select the **Shape Script** radio button and then click on the **Assign** button. The [Shape Script Editor](#) <sup>62</sup> displays.



Type the example Shape Scripts in the **Edit** window. You can click on the **Refresh** button in order to view the shape in the preview window.

**Note:**

If you define a composite Shape Script (such as the connector at the end of the [Example Scripts](#) topic), click on the **Next Shape** button to page through the components of the shape.

Once you have finished [writing your Shape Script](#), click on the **OK** button. To save the Shape Script you must click on the **Save** button on the **Stereotypes** tab.

Once you have created your Shape Script for a particular stereotype, you can assign that stereotype to an element or connector. The appearance reflects the Shape Script you created. To do this, drag and drop the appropriate element or connector into your diagram.

**Notes:**

- Shape Scripts do not function in Sequence diagrams.
- If an element's appearance is modified by a Shape Script, many of the options on the **Advanced** context menu for that element are disabled (see *Using Enterprise Architect - UML Modeling Tool*).

Right-click on the element or connector and select the **Properties** context menu option. Click on the **Stereotype** field drop-down arrow, select the stereotype you created and click on the **OK** button. The object's shape now reflects the Shape Script you created.

The screenshot shows the 'Shape Editor' dialog box with the following details:

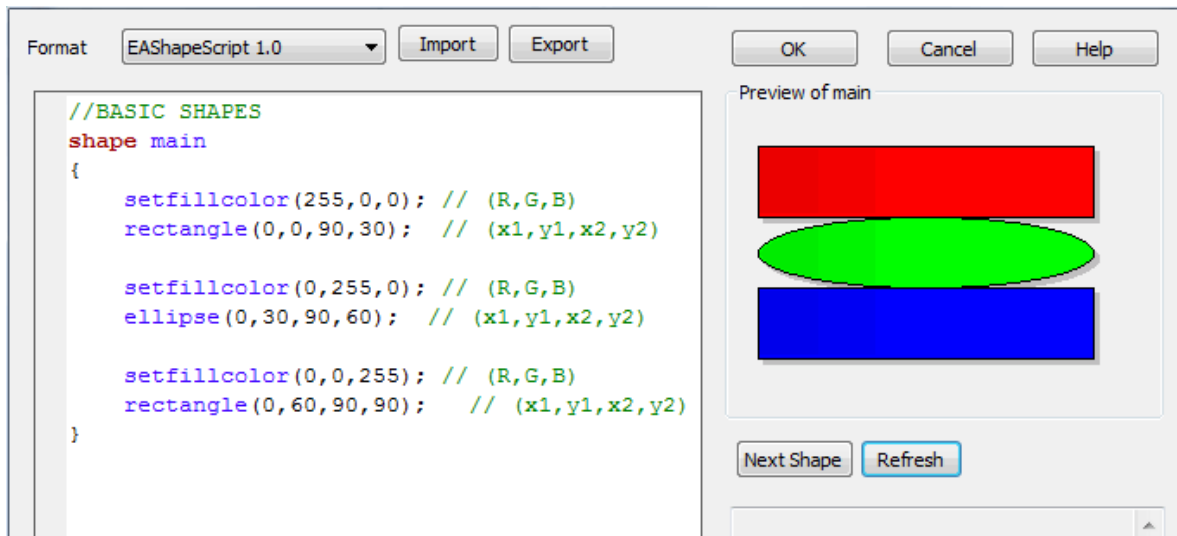
- General Tab:**
  - Name: Class 1
  - Stereotype: architecture (dropdown menu)
  - Author: Frederick Walter (dropdown menu)
  - Scope: Public (dropdown menu)
  - Alias: (empty text field)
  - Persistence: (dropdown menu)
  - Phase: 1.0 (text field)
  - Version: 1.0 (text field)
  - Status: Proposed (dropdown menu)
  - Complexity: Easy (dropdown menu)
  - Language: C++ (dropdown menu)
  - Keywords: (empty text field)
  - Abstract:
  - Advanced: (button)
- Notes:**
  - Rich text editor toolbar with icons for Bold (B), Italic (I), Underline (U), Text Color (A), Bulleted List, Numbered List, Indent, and Undo/Redo (x<sup>2</sup>, x<sub>2</sub>).
  - Large empty text area for notes.
- Buttons:** OK, Cancel, Apply, Help.

### 3.2 Shape Editor

The **Shape Editor** enables you to create [Shape Scripts](#)<sup>[59]</sup>. It provides the facilities of the *Common Code Editor*, including Intellisense for shape script attributes and functions. 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 **Shape Editor**, follow the steps below:

1. Select the **Settings | UML** menu option; the **UML Types** dialog displays, defaulted to the **Stereotypes** tab.
2. Type a name in the **Stereotype** field, or click on an the required stereotype in the list.
3. From the **Override Appearance** panel, select the **Shape Script** radio button.
4. Click on the **Assign** button. The **Shape Editor** dialog displays.



Option	Use to
<b>Format</b>	Select the Shape Script version.
<b>Import</b>	Import a Shape Script from a text file.
<b>Export</b>	Export a Shape Script to a text file.
<b>OK</b>	Exit from the <b>Shape Editor</b> , don't forget to save your script from the <b>Stereotypes</b> tab. See <a href="#">Getting Started</a> <sup>[59]</sup> .
<b>Next Shape</b>	Rotate though the multiple shape definitions.
<b>Refresh</b>	Parse your script and display the result in the <b>Preview</b> window.

### 3.3 Write Scripts

This topic is a detailed reference for writing [Shape Scripts](#) <sup>[59]</sup>. For an introduction to writing Shape Scripts, see the [Getting Started](#) <sup>[59]</sup> and [Example Scripts](#) <sup>[75]</sup> topics.

See the following reference topics for more detailed information on shape scripting:

- [Syntax grammar](#) <sup>[63]</sup>
- [Shape attributes](#) <sup>[64]</sup>
- [Drawing methods](#) <sup>[66]</sup>
- [Color queries](#) <sup>[70]</sup>
- [Conditional branching](#) <sup>[70]</sup>
- [Query methods](#) <sup>[70]</sup>
- [Display Item properties](#) <sup>[70]</sup>
- [Sub-shapes](#) <sup>[73]</sup>
- [Reserved names](#) <sup>[74]</sup>
- [Miscellaneous](#) <sup>[74]</sup>

#### 3.3.1 Syntax Grammar

Grammar symbols:

- \* = zero or more
- + = one or more

- | = or
- ; = terminator

<b>ShapeScript</b>	::=	<Shape>*;
<b>Shape</b>	::=	<ShapeDeclaration> <ShapeBody>;
<b>ShapeDeclaration</b>	::=	<ShapeType> <ShapeName>;
<b>ShapeType</b>	::=	"shape"   "decoration";
<b>ShapeName</b>	::=	<ReservedShapeName>   <stringliteral>;
<b>ReservedShapeName</b>	::=	See <a href="#">Reserved Names</a> <sup>[74]</sup> for full reserved shape listing
<b>ShapeBody</b>	::=	"{" <InitialisationAttributeAssignment>* <DrawingStatement>* <SubShape>* "}";
<b>InitialisationAttributeAssignment</b>	::=	<Attribute> "=" <Value> ",";
<b>Attribute</b>	::=	See <a href="#">Shape Attributes</a> <sup>[64]</sup> for full listing of attribute names
<b>DrawingStatement</b>	::=	<IfElseSection>   <Method>;
<b>IfElseSection</b>	::=	"if" "(" <QueryExpression> ")" <TrueSection> [<ElseSection>];
<b>QueryExpression</b>	::=	<QueryName> "(" <ParameterList> ")";
<b>QueryName</b>	::=	See <a href="#">Query Methods</a> <sup>[70]</sup> for a full listing of Query names
<b>TrueSection</b>	::=	"{" <DrawingStatement>* }"
<b>ElseSection</b>	::=	"else" "{" <DrawingStatement>* }"
<b>Method</b>	::=	<MethodName> "(" <ParameterList> ")" ";"
<b>MethodName</b>	::=	See <a href="#">Drawing Methods</a> <sup>[66]</sup> for a full listing of method names

### 3.3.2 Shape Attributes

*syntax: attribute "=" value ";"*

example:

```

shape main
{
    //Initialisation attributes - must be before drawing commands
    noshadow = "true";
    h_align = "center";

    //drawing commands
    rectangle(0,0,100,100);
    println("foo bar");
}

```

Attribute Name	Type	Description
<b>bottomAnchorOffset</b>	<i>(int,int)</i>	When creating a Shape Script for an embedded element (such as a Port), use this attribute to offset the shape from the bottom edge of its parent.  For example: <i>bottomAnchorOffset=(0,-10)</i> ; move embedded element up 10 pixels from the bottom edge

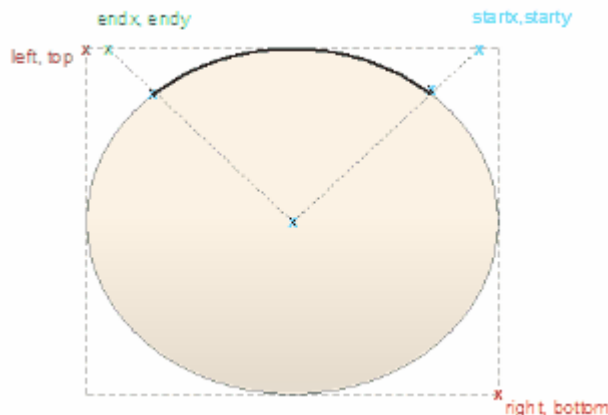


Attribute Name	Type	Description
<b>dockable</b>	<i>string</i>	Makes the shape default to dockable (see <i>Using Enterprise Architect - UML Modeling Tool</i> ), so that it can be aligned with and joined to other elements (both other Shape Scripts and standard elements) on the current diagram. You cannot reverse the dockable status with the <b>Appearance</b> menu option; to change the status, you must edit the Shape Script.  Valid values: <b>standard</b> or <b>off</b>
<b>editableField</b>	<i>string</i>	Defines a shape as an editable region of the element.  This field impacts element shapes only, line glyphs are not supported.  Valid Values: <b>alias</b> , <b>name</b> , <b>note</b> , <b>stereotype</b>
<b>endPointY,</b> <b>endPointX</b>	<i>integer</i>	Only used for the reserved target and source shapes for connectors; this point determines where the main connector line connects to the end shapes.  Default: <b>0</b> and <b>0</b>
<b>fixedAspectRatio</b>	<i>string</i>	Set to <b>true</b> to fix the aspect ratio. Do not use if you do not want to fix the aspect ratio.
<b>h_Align</b>	<i>string</i>	Affects horizontal placement of printed text and subshapes depending on the <b>layoutType</b> attribute.  Valid values: <b>left</b> , <b>center</b> , or <b>right</b>
<b>layoutType</b>	<i>string</i>	Determines how subshapes are sized and positioned. See <a href="#">Subshape Layout</a> for further details.  Valid values: <b>leftright</b> , <b>topdown</b> , <b>border</b>
<b>leftAnchorOffset</b>	<i>(int,int)</i>	When creating a Shape Script for an embedded element (such as a Port), use this attribute to offset the shape from the left edge of its parent.  For example:  <i>leftAnchorOffset=(10,0)</i> ; move embedded element right 10 pixels from the left edge
<b>noShadow</b>	<i>string</i>	Set to <b>true</b> to suppress the shapes shadow from being rendered.  Valid values: <b>true</b> or <b>false</b> (default= <b>false</b> )
<b>orientation</b>	<i>string</i>	Applies to decoration shapes only. Determines where the decoration is positioned within the containing element glyph.  Valid values: <b>NW</b> , <b>N</b> , <b>NE</b> , <b>E</b> , <b>SE</b> , <b>S</b> , <b>SW</b> , <b>W</b>
<b>preferredHeight</b>		Used by <b>border</b> layoutType - north and south  Used in drawing the source and target shapes for connectors to determine how wide the line is.
<b>preferredWidth</b>		Used by <b>border</b> layoutType - east and west.  Used by <b>leftright</b> layoutType, shapes where <i>scalable</i> is <b>false</b> to determine how much space they occupy for layout purposes.
<b>rightAnchorOffset</b>	<i>(int,int)</i>	When creating a Shape Script for an embedded element (such as a Port), use this attribute to offset the shape from the right edge of its parent.  For example:  <i>rightAnchorOffset=(-10,0)</i> ; move embedded element left 10 pixels from the right edge
<b>rotatable</b>	<i>string</i>	Set to <b>false</b> to prevent rotation of the shape. This attribute is only

Attribute Name	Type	Description
		applicable to the source and target shapes for lines glyphs. Valid values: <b>true</b> or <b>false</b> (default = <b>true</b> )
<b>scalable</b>	<i>string</i>	Set to <b>false</b> to stop the shape from being relatively sized to the associated diagram glyph. Valid values: <b>true</b> or <b>false</b> (default= <b>true</b> )
<b>topAnchorOffset</b>	<i>(int,int)</i>	When creating a Shape Script for an embedded element (such as a Port), use this attribute to offset the shape from the top edge of its parent. For example: <i>topAnchorOffset=(0,10)</i> ; move embedded element down 10 pixels from the top edge
<b>v_Align</b>	<i>string</i>	Affects vertical placement of printed text and subshapes depending on the <b>layoutType</b> attribute. Valid values: <b>top</b> , <b>center</b> , or <b>bottom</b>

### 3.3.3 Drawing Methods

Method Name	Description
<b>addsubshape</b> ( <b>string</b> shapename[, <b>int</b> width, <b>int</b> height])	Adds a sub-shape with the name <b>shapename</b> that must be defined within the current shape definition.
<b>arc</b> ( <b>int</b> left, <b>int</b> top, <b>int</b> right, <b>int</b> bottom, <b>int</b> startingpointx, <b>int</b> startingpointy, <b>int</b> endingpointx, <b>int</b> endingpointy)	Draws an elliptical anticlockwise arc with the ellipse having extents at <b>left</b> , <b>top</b> , <b>right</b> and <b>bottom</b> . The start point of the arc is defined by the intersection of the ellipse and the line from the center of the ellipse to the point ( <b>startingpointx</b> , <b>startingpointy</b> ). The end of the arc is similarly defined by the intersection of the ellipse and the line from the center of the ellipse to the point ( <b>endingpointx</b> , <b>endingpointy</b> ). For example: <code>Arc(0, 0, 100, 100, 95, 0, 5, 0);</code>
<b>arco</b> ( <b>int</b> left, <b>int</b> top, <b>int</b> right,	As for the arc method, except that a line is drawn from the current position to the starting point of the arc, and then the current position is updated to the end point of the arc.



Method Name	Description
<b>int bottom,</b> <b>int startingpointx,</b> <b>int startingpointy,</b> <b>int endingpointx,</b> <b>int endingpointy)</b>	
<b>bezierto(</b> <b>int controlpoint1x,</b> <b>int controlpoint1y,</b> <b>int controlpoint2x,</b> <b>int controlpoint2y,</b> <b>int endpointx,</b> <b>int endpointy)</b>	Draws a bezier curve and updates the pen position.
<b>defSize(int width, int height)</b>	<p>Sets the default size of the element.</p> <p>This can appear in IF and ELSE clauses with different values in each, and causes the element to be resized automatically each time the values change. For example:</p> <pre>if(HasTag("horizontal","true")) {     defSize(100,20);     rectangle(0,0,100,100); } else {     defSize(20,100);     rectangle(0,0,100,100); }</pre> <p>The above example sets the shape to the specified default size each time the Tagged Value <i>horizontal</i> is changed.</p> <p>When this is set, <b>[Alt]+[Z]</b> also resizes the shape to the defined dimensions.</p> <p><b>Note:</b></p> <p>The minimum value for both <b>int width</b> and <b>int height</b> is 10.</p>
<b>drawnativeshape()</b>	<p>Causes Enterprise Architect to render the shape using its usual, non-Shapescript notation. Subsequent drawing commands are super-imposed over the native notation.</p> <p>This method is only enabled for element Shape Scripts; line Shape Scripts are not supported.</p>
<b>ellipse(</b> <b>int left,</b> <b>int top,</b> <b>int right,</b> <b>int bottom)</b>	Draws an ellipse with extents defined by <b>left</b> , <b>top</b> , <b>right</b> and <b>bottom</b> .
<b>endpath()</b>	Ends the sequence of drawing commands that define a path.
<b>fillandstrokepath()</b>	Fills the previously defined path with the current fill color, then draws its outline with the current pen.
<b>fillpath()</b>	Fills the previously defined path with the current fill color.
<b>hidelabel(</b> <b>string labelname)</b>	Hides the label specified by <b>labelname</b> .
<b>image(</b> <b>string imageld,</b>	Draws the image that has the name <b>imageld</b> in the <b>Image Manager</b> .

Method Name	Description
<b>int left,</b> <b>int top,</b> <b>int right,</b> <b>int bottom)</b>	<p><b>Note:</b></p> <p>The image must exist within the model in which the stereotype is used. If it does not already exist in the model, you must import it as reference data (see <i>UML Model Management</i>).</p>
<b>lineto(</b> <b>int x,</b> <b>int y)</b>	<p>Draws a line from the current cursor position to a point specified by <b>x</b> and <b>y</b>, and then updates the pen cursor to that position.</p>
<b>moveto(</b> <b>int x,</b> <b>int y)</b>	<p>Moves the pen cursor to the point specified by <b>x</b> and <b>y</b>.</p>
<b>polygon(</b> <b>int centerx,</b> <b>int centery,</b> <b>int numberofsides,</b> <b>int radius,</b> <b>float rotation)</b>	<p>Draws a regular polygon with center at the point (<b>centerx</b>, <b>centery</b>), and <b>numberofsides</b> number of sides.</p>
<b>print(</b> <b>string text)</b>	<p>Prints the specified text string.</p> <p><b>Note:</b></p> <p>You cannot change the font size, type or color of this text.</p>
<b>printifdefined(</b> <b>string</b> <b>propertyname,</b> <b>string truepart[,</b> <b>string falsepart])</b>	<p>Prints the <i>truepart</i> if the given property exists and has a non-empty value, otherwise prints the optional <i>falsepart</i>.</p> <p><b>Note:</b></p> <p>You cannot change the font size, type or color of this text.</p>
<b>println(</b> <b>string text)</b>	<p>Appends a line of text to the shape and a line break.</p> <p><b>Note:</b></p> <p>You cannot change the font size, type or color of this text.</p>
<b>printwrapped(</b> <b>string text)</b>	<p>Prints the specified text string, wrapped over multiple lines if the text is wider than its containing shape.</p> <p><b>Note:</b></p> <p>You cannot change the font size, type or color of this text.</p>
<b>rectangle(</b> <b>int left,</b> <b>int top,</b> <b>int right,</b> <b>int bottom)</b>	<p>Draws a rectangle with extents at <b>left</b>, <b>top</b>, <b>right</b>, <b>bottom</b>. Values are percentages.</p>
<b>roundrect(</b> <b>int left,</b> <b>int top,</b> <b>int right,</b> <b>int bottom,</b> <b>int</b> <b>abs_cornerwidth,</b> <b>int</b>	<p>Draws a rectangle with rounded corners, with extents defined by <b>left</b>, <b>top</b>, <b>right</b> and <b>bottom</b>. The size for the corners is defined by <b>abs_cornerwidth</b> and <b>abs_cornerheight</b>; these values do not scale with the shape.</p>

Method Name	Description
<code>abs_cornerheight)</code>	
<code>setdefaultcolors()</code>	Returns the brush and pen color to the default settings, or to the user-defined colors if available. See <a href="#">Color Queries</a> [70].
<code>setfillcolor(</code> <code>int red,</code> <code>int green,</code> <code>int blue)</code>  <code>setfillcolor(</code> <code>Color newColor)</code>	Sets the fill color.  You can specify the required color by defining RGB values or using a color value returned by any of the <a href="#">Color Queries</a> [70]; for example: <code>GetUserFillColor()</code>
<code>setlinestyle(</code> <code>string linestyle)</code>	Changes the stroke pattern for commands that use the pen. Parameters: <b>string linestyle:</b> the following styles are valid: <ul style="list-style-type: none"> <li>• <b>solid</b></li> <li>• <b>dash</b></li> <li>• <b>dot</b></li> <li>• <b>dashdot</b></li> <li>• <b>dashdotdot</b></li> </ul>
<code>setorigin(</code> <code>string relativeTo,</code> <code>int xOffset,</code> <code>int yOffset)</code>	Positions floating text labels relative to the main shape. <b>relativeTo</b> is one of <b>N, NE, E, SE, S, SW, W, NW, CENTER</b> <b>xOffset</b> and <b>yOffset</b> are in pixels, not percentage values, and can be negative.
<code>setpen(</code> <code>int red,</code> <code>int green,</code> <code>int blue[,</code> <code>int penwidth])</code>	Sets the pen to the defined color and optionally sets the pen width.  <b>Note:</b> This method is only for line-drawing commands. It does not affect any text commands.
<code>setpencolor(</code> <code>int red,</code> <code>int green,</code> <code>int blue)</code>  <code>setpencolor(</code> <code>Color newColor)</code>	Sets the pen color. You can specify the required color by defining RGB values or using a color value returned by any of the <a href="#">Color Queries</a> [70]; for example: <code>GetUserFillColor()</code>  <b>Note:</b> This method is only for line-drawing commands. It does not affect any text commands.
<code>setpenwidth(</code> <code>int penwidth)</code>	Sets the width of the pen. Pen width should be between <b>1</b> and <b>5</b> .  <b>Note:</b> This method is only for line-drawing commands. It does not affect any text commands.
<code>showlabel(</code> <code>string labelname)</code>	Reveals the hidden label specified by <b>labelname</b> .
<code>startcloudpath(</code> <code>puffWidth,</code> <code>puffHeight,</code> <code>noise)</code>	Similar to <b>StartPath</b> , except that it draws the path with cloud-like curved segments ( <i>puffs</i> ). Parameters:

Method Name	Description
	<ul style="list-style-type: none"> <li>• <i>float</i> <b>puffWidth</b> (default = <b>30</b>), the horizontal distance between puffs</li> <li>• <i>float</i> <b>puffHeight</b> (default = <b>15</b>), the vertical distance between puffs</li> <li>• <i>float</i> <b>noise</b> (default = <b>1.0</b>), the randomization of the puffs' positions.</li> </ul>
<b>startpath()</b>	Starts the sequence of drawing commands that define a path.
<b>strokepath()</b>	Draws the outline of the previously defined path with the current pen.

### 3.3.4 Color Queries

Color queries can only be used to retrieve arguments for the *SetPenColor* and *SetFillColor* commands. These queries can be used in place of the arguments.

```

getUserFillColor()
getUserBorderColor()
getUserFontColor()
getUserPenSize()
    shape main
    {
        setfillcolor(getuserbordercolor());
        setpencolor(getuserfillcolor());

        rectangle(0,0,100,100);
    }

```

### 3.3.5 Conditional Branching

Shape Scripts provide condition branching with the *if else* statement, and query methods that evaluate to either *True* or *False*. See:

- [Syntax Grammar](#)<sup>[63]</sup> for IF statement syntax.
- [Query Methods](#)<sup>[70]</sup> for methods that can be used as the conditional expression for IF statements.
- [Example Scripts](#)<sup>[75]</sup> for an example.

### 3.3.6 Query Methods

Two query methods are available for seeing if the associated element has certain tags or properties; these methods can be used as the conditional expression for an *if else* statement.

Method	Description
<b>boolean HasTag( string tagname, [string tagvalue])</b>	Returns <b>true</b> if the associated element has a tag value with the name <i>tagname</i> . If the second parameter <i>tagvalue</i> is provided, the tag <i>tagname</i> must be present, and the value of the tag has to be equal to <i>tagvalue</i> for the method to return <b>true</b> .
<b>boolean HasProperty( string propertyname, [string propertyvalue])</b>	Returns <b>true</b> if the associated element has a property with the name <i>propertyname</i> . If the second parameter <i>propertyvalue</i> is provided, the property must be present, and the value of the property has to be equal to <i>propertyvalue</i> for the method to return <b>true</b> . See <a href="#">Display Item Properties</a> <sup>[70]</sup> for a list of valid values for <i>propertyname</i> .

### 3.3.7 Display Item Properties

The commands **print**, **println**, and **printwrapped** all take a string parameter representing the text to be printed. Element and connector properties can be added to the text using the substitution macro *#propertyname#*.

For example: `println("name: #NAME#");`

In addition to the properties listed below, Tagged Values can also be displayed by prefixing the Tagged Value name with TAG:.

For example: `print("#TAG:condition#");`

## Properties Visible to Shape Scripts

### Properties for Element Shape Scripts

- alias
- author
- cardinality
- classifier
- classifier.alias
- classifier.metatype
- classifier.name
- classifier.stereotype
- complexity
- concurrency
- datecreated
- datemodified
- diagram.mdgtype
- diagram.name
- diagram.stereotype
- diagram.type
- haslinkedddocument
- isabstract
- isactive
- iscomposite
- isembedded
- isinparent
- isleaf
- islocked
- isroot
- isspec
- istagged
- keywords
- language
- metatype
- multiplicity
- name
- notes
- packagename
- parentedge
- parent.metatype
- persistence
- phase
- propertytype
- propertytype.alias
- propertytype.metatype
- propertytype.name
- propertytype.stereotype

- scope
- status
- stereotype
- type
- version
- visibility.

#### **Properties for Connector Shape Scripts**

- alias
- diagram.connectornotation
- diagram.mdgtype
- diagram.name
- diagram.stereotype
- diagram.type
- direction
- isroot
- isleaf
- name
- notes
- source.aggregation
- source.alias
- source.changeable
- source.constraints
- source.element.name
- source.element.stereotype
- source.metatype
- source.multiplicity
- source.multiplicityisordered
- source.qualifiers
- source.stereotype
- source.targetscope
- stereotype
- subtype
- target.aggregation
- target.alias
- target.changable
- target.constraints
- target.element.name
- target.element.stereotype
- target.metatype
- target.multiplicity
- target.multiplicityisordered
- target.qualifiers
- target.stereotype
- target.targetscope
- type.



### 3.3.8 Sub-Shapes

Shapes can contain - and be composed of - other shapes.

#### Subshape Layout

To set the layout type, the *layoutType* attribute must be set in the **initialization attributes** section of the script; in other words, before any of the methods are called. Valid values for this attribute are:

##### LeftRight

Shapes with *leftright* position subshapes side by side, with the first added on the left, and subsequent subshapes to the right.

##### TopDown

*TopDown* places subshapes in a vertical arrangement, with the first shape added to the top and subsequent shape added below.

##### Border

*Border* layout requires an additional argument to the *addsubshape* method to specify which region of the containing shape the subshape is to occupy: *N*, *E*, *S*, *W* or *CENTER*. Each region can only be occupied by one subshape.

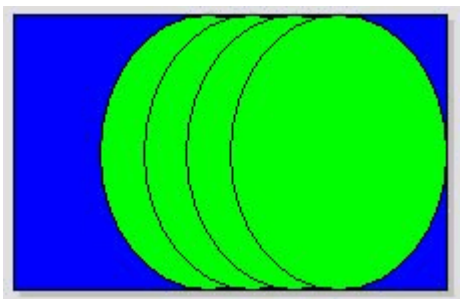
A subshape that is assigned to the *E* or *W* region must have its *preferredwidth* attribute specified in its declaration. Similarly, subshapes added to *N* or *S* must have their *preferredheight* attribute set. In this case, the values for these attributes are treated as static lengths and do not scale glyphs.

For example:

```
shape main
{
  layouttype="topdown";
  setfillcolor{0,0,255};
  rectangle{0,0,100,100};
  addsubshape("sub",50,100,20,0);
  addsubshape("sub",50,100,30,-100);
  addsubshape("sub",50,100,40,-200);
  addsubshape("sub",50,100,50,-300);

  shape sub
  {
    setfillcolor{0,255,0};
    ellipse{0,0,100,100};
  }
}
```

The above script provides the following shape:



### 3.3.9 Reserved Names

#### Elements

Elements (such as Class, State or Event) have the following reserved names for parts of the shape.

Name	Description
<b>shape main</b>	The <b>main</b> shape is the whole shape.
<b>shape label</b>	The shape <b>label</b> gives the shape a detached label.
<b>decoration</b> <identifier>	<b>Decoration</b> gives the shape a decoration as defined by the name in <identifier>.

#### Connectors

Connectors (such as Association, Dependency or Generalization) have the following reserved names for parts of the shape.

Name	Description
<b>shape main</b>	The <b>main</b> shape is the whole shape.
<b>shape source</b>	The <b>source</b> shape is an extra shape at the source end of the connector.
<b>shape target</b>	The <b>target</b> shape is an extra shape at the target end of the connector.
<b>shape &lt;labelID&gt;</b>	The <labelID> gives the connector a detached label, where <labelID> is one of the following: <ul style="list-style-type: none"> <li>• <b>LeftTopLabel</b></li> <li>• <b>MiddleTopLabel</b></li> <li>• <b>RightTopLabel</b></li> <li>• <b>LeftBottomLabel</b></li> <li>• <b>MiddleBottomLabel</b></li> <li>• <b>RightBottomLabel</b></li> </ul>

### 3.3.10 Miscellaneous

#### Return Command

Execution of the script can be terminated by using the return command. Please see [Example Scripts](#) <sup>75</sup> for an example.

#### Looping

The Shape Script feature does not support looping constructs.

#### Comments

C-style comments are supported. For example:

```
// C Style Single Line comment
/* Multi Line
comment supported */
```

#### String Manipulation

Not Supported.

### Arithmetical Operations

Not Supported.

### Variables Declaration

Not Supported.

### Change ShapeScript Fonts

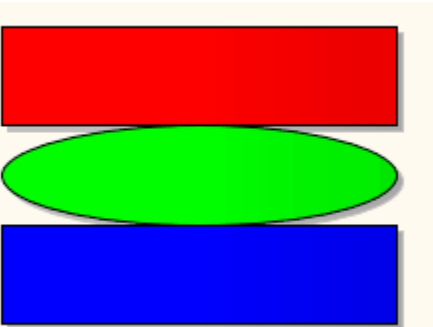
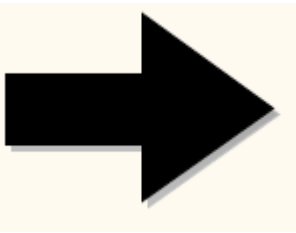

Not possible.

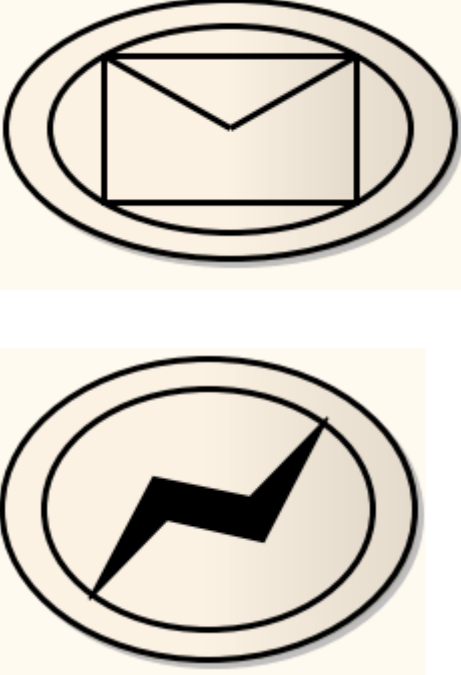
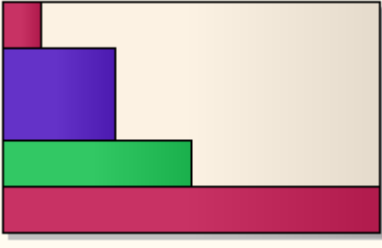
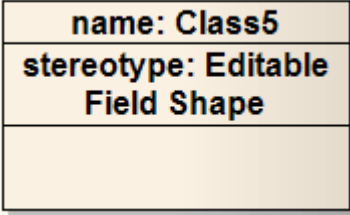
### Can I apply a Shapascript without using Stereotypes?

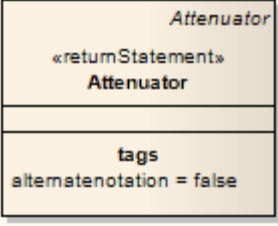
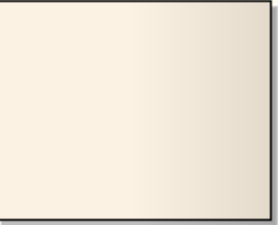
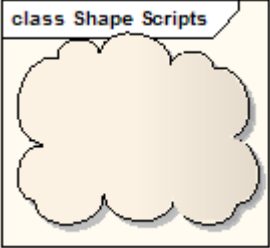
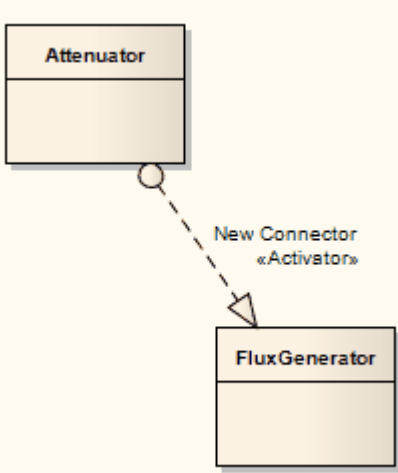
No.

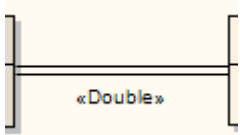
## 3.4 Example Scripts

Below is a selection of example Shape Scripts.

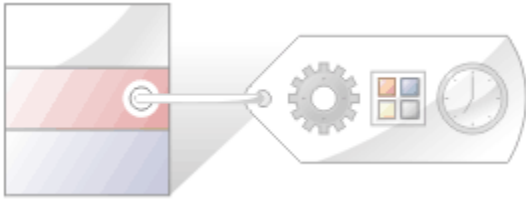
Code	Result
<pre>//BASIC SHAPES shape main {     setfillcolor(255,0,0); // (R,G,B)     rectangle(0,0,90,30); // (x1,y1,x2,y2)      setfillcolor(0,255,0); // (R,G,B)     ellipse(0,30,90,60); // (x1,y1,x2,y2)      setfillcolor(0,0,255); // (R,G,B)     rectangle(0,60,90,90); // (x1,y1,x2,y2) }</pre>	
<pre>//SINGLE CONDITIONAL SHAPE shape main {     if (HasTag("Trigger","Link"))     { // Only draw if the object has a Tagged Value       // Trigger=Link       // Set the fill color for the path       setfillcolor(0,0,0);       startpath(); // Start to trace out a path       moveto(23,40);       lineto(23,60);       lineto(50,60);       lineto(50,76);       lineto(76,50);       lineto(50,23);       lineto(50,40);       endpath(); // End tracing out a path       // Fill the traced path with the fill color       fillandstrokepath();       return;     } }</pre>	
<pre>//MULTI CONDITIONAL SHAPE shape main {     startpath();     ellipse(0,0,100,100);     endpath();     fillandstrokepath();     ellipse(3,3,27,27); }</pre>	

Code	Result
<pre> if (HasTag("Trigger","None")) {     return; }  if (HasTag("Trigger","Error")) {     setfillcolor(0,0,0);     startpath();     moveto(23,77);     lineto(37,40);     lineto(60,47);     lineto(77,23);     lineto(63,60);     lineto(40,53);     lineto(23,77);     endpath();     fillandstrokepath();     return; }  if (HasTag("Trigger","Message")) {     rectangle(22,22,78,78);     moveto(22,22);     lineto(50,50);     lineto(78,22);     return; } </pre>	
<pre> //SUB SHAPES shape main {     rectangle(0,0,100,100);      addsubshape("red", 10, 20);     addsubshape("blue", 30, 40);     addsubshape("green", 50, 20);     addsubshape("red", 100, 20);      shape red     {         setfillcolor(200, 50, 100);         rectangle(0,0,100,100);     }      shape blue     {         setfillcolor(100, 50, 200);         rectangle(0,0,100,100);     }      shape green     {         setfillcolor(50, 200, 100);         rectangle(0,0,100,100);     } } </pre>	
<pre> //Editable Field Shape shape main {     rectangle(0,0,100,100);      addsubshape("namecompartment", 100, 20);     addsubshape("stereotypecompartment", 100, 40);      shape namecompartment     {         h_align = "center";         editablefield = "name";          rectangle(0,0,100,100);         println("name: #name#");     } } </pre>	

Code	Result
<pre> shape stereotypecompartment {     h_align = "center";     editablefield = "stereotype";      rectangle(0,0,100,100);     println("stereotype: #stereotype#"); }         </pre>	
<pre> //Return Statement Shape shape main {     if(hasTag("alternatenotation", "false"))     {         //draw ea's inbuild glyph         drawnativeshape();         //exit script with the return statement         return;     }      //alternate notation commands     //...     rectangle(0,0,100,100); }         </pre>	 
<pre> //Cloud Path Example Shape shape main {     StartCloudPath();     Rectangle(0,0,100,100);     EndPath();     FillAndStrokePath(); }         </pre>	
<pre> // Connector Example shape main {     // draw a dashed line     noshadow=true;     setlinestyle("DASH");     moveto(0,0);     lineto(100,0); }  shape source {     // draw a circle at the source end     rotatable = true;     startpath();     ellipse(0,6,12,-6);     endpath();     fillandstrokepath(); }  shape target {     // draw an arrowhead at the target end     rotatable = true;     startpath();     moveto(0,0);     lineto(16,6); }         </pre>	

Code	Result
<pre>lineto(16,-6); endpath(); fillandstrokepath(); }</pre>	
<pre>// Double Line shape main {     noshadow=true;     moveto(0,-10);     lineto(100,-10);     moveto(0,10);     lineto(100,10); }</pre>	

## 4 Tagged Value Types



Enterprise Architect provides a number of predefined Tagged Value Types that enable you to create your own:

- Tagged Values that are [structured](#)<sup>[79]</sup> with a specific format, with or without tag [filters](#)<sup>[80]</sup>, or
- Tagged Values that return values from the various [reference data](#)<sup>[82]</sup> tables.

You can also use a masking parameter to create your own customized [masked Tagged Value Type](#)<sup>[84]</sup>.

### Note:

You can transport Tagged Value Type definitions between models, using the **Export Reference Data** and **Import Reference Data** options on the **Tools** menu. Tagged Value Types are exported as *Property Types*. See *UML Model Management*.


### 4.1 Predefined Structured Types

This table details the predefined structured Tagged Value types, along with the syntax used to create the initial values for their use. You use these to [create your own structured Tagged Values](#)<sup>[81]</sup>.

### Note:

**Tagged Value Type** and **Format** entries are case-sensitive.

Tagged Value Type	Format	Description
Boolean	<b>Type=Boolean;</b>	Enables input of a true or false value.
Classifier	<b>Type=Classifier; Values=Type1,Type2; Stereotypes=Stereotype 1;</b>	<b>Deprecated. Use RefGUID and RefGUIDList.</b> Returns the <i>name</i> of a user-selected element from the model, where <b>Type1</b> and <b>Type2</b> specify one or more allowed element types and <b>Stereotype1</b> represents an allowed stereotype.
Color	<b>Type=Color;</b>	Enables input of a color value from a color chooser menu.
Const	<b>Type=Const; Default=Val;</b>	Enables creation of a read-only constant value.
Custom	<b>Type= Custom;</b>	Enables you to create your own template for predefined types; more information is provided in the <a href="#">Create Custom Tagged Value Type</a> <sup>[84]</sup> topic.
DateTime	<b>Type=DateTime;</b>	Enables input of the date and time for the Tagged Value from a calendar menu.
Directory	<b>Type=Directory;</b>	Enables entry of a directory path from a browser.
Enum	<b>Type=Enum; Values=Val1,Val2,Val3; Default=Val2;</b>	Enables definition of a comma-separated list, where <b>Val1</b> , <b>Val2</b> and <b>Val3</b> represent values in the list and <b>Default</b> represents the default value of the list.

Tagged Value Type	Format	Description
File	<b>Type=File;</b>	Enables input of a filename from a file browser dialog. The named file can be launched in its default application.
Float, Decimal, Double	<b>Type=Float;</b> <b>Type=Decimal;</b> <b>Type=Double;</b>	Enable entry of a Float, Decimal or Double value. These types all map to the same type of data.
Integer	<b>Type=Integer;</b>	Enables entry of an Integer value.
Memo	<b>Type=Memo;</b>	Enables input of large and complex Tagged Values.
RefGUID	<b>Type=RefGUID;</b> <b>Values=Type1,Type2;</b> <b>Stereotypes=Stereotype 1;</b>	Enables the Tagged Value to reference an element from the model by specifying the element's <i>GUID</i> , where <b>Type1</b> and <b>Type2</b> specify one or more allowed diagram objects (such as <b>Class</b> , <b>Component</b> , <b>Attribute</b> or <b>Operation</b> ) and <b>Stereotype1</b> represents an allowed stereotype.  Set the classifier (see <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i> ), attribute or operation (see <i>The UML Dictionary</i> ) for a Tagged Value of this type by clicking on the [ ... ] button against the Tagged Value in the <b>Tagged Value</b> window.
RefGUIDList	<b>Type=RefGUIDList;</b> <b>Values=Type1,Type2;</b> <b>Stereotypes=Stereotype 1;</b>	Enables the Tagged Value to reference a list of elements from the model by specifying each element's <i>GUID</i> , where <b>Type1</b> and <b>Type2</b> specify one or more allowed diagram objects (such as <b>Class</b> or <b>Component</b> ) and <b>Stereotype1</b> represents an allowed stereotype.  Set the classifier for a Tagged Value of this type (see <i>UML Modeling With Enterprise Architect - UML Modeling Tool</i> ) by clicking on the [ ... ] button against the Tagged Value in the <b>Tagged Value</b> window.
Spin	<b>Type=Spin;</b> <b>LowerBound=x;</b> <b>UpperBound=x;</b>	Enables creation of a spin control with the value of <b>LowerBound</b> being the lowest value and <b>UpperBound</b> being the highest value.
String	<b>Type=String;</b>	Enables entry of a string value, up to 255 characters in length.  For longer texts, use <i>Type=Memo</i>  .

## Tag Filters

The following table details filters that can be used to restrict where a Tagged Value can be applied.

Filter	Format	Description
AppliesTo	<b>AppliesTo=Type1, Type2;</b>	Restricts the element types this filter can be applied to, where <b>Type1</b> and <b>Type2</b> are the valid types.  Possible values are: <ul style="list-style-type: none"> <li>• all element types</li> <li>• all connector types</li> <li>• Attribute</li> <li>• Operation and</li> <li>• OperationParameter.</li> </ul>



Filter	Format	Description
BaseStereotype	<b>BaseStereotype=S1,S2;</b>	Restricts the stereotypes that this tag belongs to, where <b>S1</b> and <b>S2</b> are the allowed stereotypes.

## 4.2 Create Structured Tagged Values

To create your own Tagged Value based on a predefined [structured Tagged Value Type](#)<sup>[79]</sup>, follow the steps below:

1. Select the **Settings | UML** menu option. The **UML Types** dialog displays. Select the **Tagged Value Types** tab.

The screenshot shows the 'UML Types' dialog with the 'Tagged Value Types' tab selected. The 'Tag Name' field contains 'Handicap' and the 'Description' field contains 'Spin Control'. The 'Detail' field contains the text: 'Type=Spin; LowerBound=0; UpperBound=250;'. Below the detail field are 'New', 'Save', and 'Delete' buttons. At the bottom is a 'Defined Tag Types' list with columns 'Type' and 'Description'. The list includes: Activity, Assignments (Assignments), Author (Returns Authors of the Model), Categories (Categories), Datafield (Database field), go (ki), Handicap (Spin Control), InMessage, InputPropertyMaps (InputPropertyMaps), and IORules (IORules). The 'Handicap' entry is highlighted in blue. At the bottom right are 'Close' and 'Help' buttons.

2. Click on the **New** button.
3. In the **Tag Name** field type an appropriate Tagged Value name.
4. In the **Description** field type the purpose of the Tagged Value, if required.
5. In the **Detail** field copy-and-paste or type the syntax of the predefined structured Tagged Value Type.

In the example above (which is used in the definition of a field to enable a user to set a sports handicap) the predefined type is **Spin**, with the Upper and Lower Bounds set for the field values. (*Spin* is the Microsoft term for selection arrows in a variable field - the user clicks on the arrows to increase or decrease the value between the upper and lower bound limits.)

6. Click on the **Save** button.

The Tagged Value type displays in the **Defined Tag Types** list.

### 4.3 Predefined Reference Data Types

This table details the predefined Reference Data Tagged Value types that are used to return the values held in a relevant table in Enterprise Architect, along with the syntax required for their use. You use these to [create your own Reference Data Tagged Values](#) <sup>[83]</sup>.

Tagged Value Type	Format	Drop-Down List Returned
Authors	<b>Type=Enum; List=Authors;</b>	Authors that have been defined for the Enterprise Architect model.
Cardinality	<b>Type=Enum; List=Cardinality;</b>	Cardinality types that have been defined for the Enterprise Architect model.
Clients	<b>Type=Enum; List=Clients;</b>	Clients that have been defined for the Enterprise Architect model.
ComplexityTypes	<b>Type=Enum; List=ComplexityTypes;</b>	Complexity types that have been defined for the Enterprise Architect model.  Whilst complexity types can be exported and imported as project reference data, they cannot be updated and so are effectively standard across all projects.
ConstraintTypes	<b>Type=Enum; List=ConstraintTypes;</b>	Constraint types that have been defined for the Enterprise Architect model.
EffortTypes	<b>Type=Enum; List=EffortTypes;</b>	Effort types that have been defined for the Enterprise Architect model.
MaintenanceTypes	<b>Type=Enum; List=MaintenanceTypes ;</b>	Maintenance types that have been defined for the Enterprise Architect model.
ObjectTypes	<b>Type=Enum; List=ObjectTypes;</b>	Object types that have been defined for the Enterprise Architect model.
Phases	<b>Type=Enum; List=Phases;</b>	Phases that have been defined for the Enterprise Architect model.
ProblemTypes	<b>Type=Enum; List=ProblemTypes;</b>	Problem types that have been defined for the Enterprise Architect model.
RoleTypes	<b>Type=Enum; List=RoleTypes;</b>	Role types that have been defined for the Enterprise Architect model.
RequirementTypes	<b>Type=Enum; List=RequirementTypes ;</b>	Requirement types that have been defined for the Enterprise Architect model.
Resources	<b>Type=Enum; List=Resources;</b>	Resources that have been defined for the Enterprise Architect model.
RiskTypes	<b>Type=Enum; List=RiskTypes;</b>	Risk types that have been defined for the Enterprise Architect model.
RTFTemplates	<b>Type=Enum; List=RTFTemplates;</b>	RTF Templates that have been defined for the Enterprise Architect model.
ScenarioTypes	<b>Type=Enum; List=ScenarioTypes;</b>	Scenario types that have been defined for the Enterprise Architect model.
TestTypes	<b>Type=Enum; List=TestTypes;</b>	Test types that have been defined for the Enterprise Architect model.

## 4.4 Create Reference Data Tagged Values

To create your own Tagged Value based on a [predefined Reference Data Tagged Value Type](#)<sup>[82]</sup>, follow the steps below:

1. Select the **Settings | UML** menu option. The **UML Types** dialog displays; select the **Tagged Value Types** tab.

The screenshot shows the 'UML Types' dialog with the 'Tagged Value Types' tab selected. The 'Tag Name' field contains 'Author' and the 'Description' field contains 'Returns Authors of the Model'. The 'Detail' field contains the text: 'Type=Enum; Type=List; List=Authors;'. Below the fields are three buttons: 'New', 'Save', and 'Delete'. At the bottom, there is a table titled 'Defined Tag Types'.

Type	Description
Activity	
Assignments	Assignments
Author	Returns Authors of the Model
Categories	Categories
Datafield	Database field

2. In the **Tag** field type an appropriate Tagged Value name.
3. In the **Description** field type a description of the purpose of the Tagged Value, if required.
4. In the **Detail** field copy-and-paste or type the syntax of the predefined Reference Data Tagged Value Type. In the example above, the Tagged Value returns the values for all of the Authors in the Enterprise Architect model.

This enables you to assign any of the previously defined Authors to a model feature (model features that can have Tagged Values applied to them are detailed in the *Model Elements and Features with Tagged Values* topic in *Using Enterprise Architect - UML Modeling Tool*).

The screenshot shows the 'Tagged Values' dialog for the 'Computer (Class)' feature. The 'Author' tag is selected, and its value is 'John Redfem'. Other tags and their values are listed in the table below.

Tag	Value
Author	John Redfem
Handicap	11
MemberAge	35
MemberPhone	(03)1234 5678
MembershipExpires	30/Dec/2099

Below the table, the 'Author' tag is highlighted in a separate section.

**Note:**

If the values in the reference data are changed after the Tagged Value Type is created, Enterprise Architect must be reloaded in order to reflect the changes in the Tagged Value Type.

## 4.5 Create Custom Tagged Value Type

Creating a custom masked Tagged Value gives you great flexibility in designing model components that accept data entries. To create a masked Tagged Value follow the steps below:

1. Select the **Settings | UML** menu option. The **UML Types** dialog displays; select the **Tagged Value Types** tab.
2. In the **Tag** field type an appropriate name for the Tagged Value.
3. In the **Description** field type the purpose of the Tagged Value, if required.
4. In the **Detail** field type **Type=Custom**;

The type **Custom** enables you to set up the appropriate mask, using the following characters to define the format of the mask:

Mask	Description
<b>D</b>	Enables the Tagged Value to display digits only.
<b>d</b>	Enables the Tagged Value to display digits or spaces.
<b>+</b>	Enables the use of <b>+</b> , <b>-</b> or <i>spaces</i> .
<b>C</b>	Enables the use of alpha characters only.
<b>c</b>	Enables the Tagged Value to be an alpha character or a space.
<b>A</b>	Enables the use of alphanumeric characters.
<b>a</b>	Enables the Tagged Value to use alphanumeric values or a space.

In the diagram below the **Mask** configuration option shows syntax that first defines seven blank spaces, which are occupied by characters determined by the template option. The first two visible characters in the **Mask** option are represented by a lower case **c** indicating that the enableable information can entered as either an alpha character or as a space. The following blank spaces again indicate space defined by the template option and the remaining characters are defined by the **d** character, which represents the enableable characters as digits or spaces. The hyphen is present in the final output, splitting up the digits.

With the **Template** configuration option, the syntax defines the template of the masked option by occupying the blank spaces that are present in the **Mask** option. The template is used to ensure that this information is present with every use of this custom Tagged Value. The underscored values indicate the area that is to be occupied by data input by the user as defined in the **Mask** option.

Stereotypes
Tagged Value Types
Cardinality Values

Tag Name:       Description:

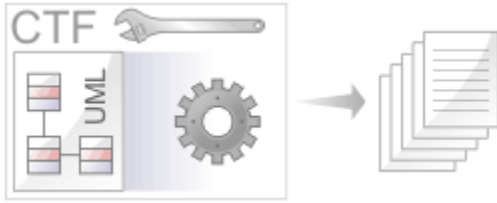
Detail:

Type=Custom:  
 Mask= cc dddd.dddd;  
 Template=State: \_\_Zip: \_\_\_\_\_-\_\_\_\_;

Defined Tag Types:

Type	Description
IORules	IORules
MemberZip	Zip Code
Message	
OutMessage	

## 5 Code Template Framework in SDK



The Code Template Framework (CTF) is used during forward engineering of UML models. It is introduced in *Code Engineering Using UML Models*. This section of the SDK discusses how you customize the way in which Enterprise Architect generates source code, using the [Code Template Editor](#).<sup>[116]</sup>

Enterprise Architect's code templates 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](#)<sup>[86]</sup> that shares some aspects of both mark-up languages and scripting languages. The Base Templates provided in Enterprise Architect are described in the *Base Templates* topic of *Code Engineering Using UML Models*.

### 5.1 Code Template Syntax

Code Templates are written as plain text, using Enterprise Architect's code template editor (see [The Code Template Editor](#)<sup>[116]</sup>). The template syntax centers on three basic constructs:

- [Literal Text](#)<sup>[86]</sup>
- [Macros](#)<sup>[87]</sup>
- [Variables](#)<sup>[115]</sup>

Templates can contain any or all of these constructs.

#### 5.1.1 Literal Text

All text within a given template that is not part of a macro or a variable definition/reference, is considered literal text. With the exception of blank lines, which are ignored, literal text is directly substituted from the template into the generated code.

Consider the following excerpt from the java Class Declaration template:

```
%PI=" "%
%CONVERT_SCOPE(classScope)%

%classStereotype=="static" ? "static": ""%
%classStereotype=="final" ? "final": ""%
%classStereotype=="static final" ? "static final": ""%
%classAbstract=="T" ? "abstract": ""%
%PI=""%
class %className%$bases
```

On the final line, the word *class*, including the subsequent space, would be treated as literal text and thus reproduced in the output. The blank line following the *CONVERT\_SCOPE* macro, however, would have no effect on the output.

The %, \$ and " characters have special meaning in the template syntax and cannot always be used as literal text. If these characters must be generated from within the templates, they can be safely reproduced using the following direct substitution macros:

Macro	Use to
<code>%d%</code>	Produce a literal \$ character.
<code>%pc%</code>	Produce a literal % character.
<code>%qt%</code>	Produce a literal " character.

## 5.1.2 Macros

Macros provide access to element fields within the UML model and are also used to structure the generated output. All macros are enclosed within percent (%) signs. The CTF contains six basic types of macros:

- [Template substitution macros](#) <sup>[87]</sup>
- [Field substitution macros](#) <sup>[88]</sup>
- [Tagged Value substitution macros](#) <sup>[100]</sup>
- [Control macros](#) <sup>[104]</sup>
- [Function macros](#) <sup>[107]</sup>
- [EASL code generation macros](#) <sup>[107]</sup>

In general, macros (including the % delimiters) are substituted with literal text in the output. For example consider the following item from the *Class Declaration* template:

```
... class %className% ...
```

The field substitution macro, *%className%*, would result in the current Class name being substituted in the output. So if the Class being generated was named *Foo*, the output would be:

```
... class Foo ...
```

### 5.1.2.1 Template Substitution Macros

*Template substitution macros* correspond to Base templates - see *Code Engineering Using UML Models*. These macros result in the execution of the named template. By convention, template macros are named according to Pascal casing.

*Structure: %<TemplateName>%*

where *<TemplateName>* can be one of the templates listed below.

When a template is referenced from within another template, it is generated with respect to the elements currently in scope. The specific template is selected based on the stereotypes of the elements in scope.

As noted previously, there is an implicit hierarchy among the various templates. Some care should be taken in order to preserve a sensible hierarchy of template references. For example, it does not make sense to use the *%ClassInherits%* macro within any of the attribute or operation templates. Conversely, the *Operation* and *Attribute* templates are designed for use within the *ClassBody* template.

The CTF contains the following template substitution macros:

- |                        |                              |                            |
|------------------------|------------------------------|----------------------------|
| • AttributeDeclaration | • ClassParameter             | • NamespaceBody            |
| • AttributeNotes       | • File                       | • NamespaceDeclaration     |
| • Attribute            | • FileImpl                   | • NamespaceImpl            |
| • Class                | • ImportSection              | • Operation                |
| • ClassImpl            | • ImportSectionImpl          | • OperationBody            |
| • ClassBase            | • InnerClass                 | • OperationBodyImpl        |
| • ClassBody            | • InnerClassImpl             | • OperationDeclaration     |
| • ClassBodyImpl        | • LinkedAttribute            | • OperationDeclarationImpl |
| • ClassDeclaration     | • LinkedAttributeNotes       | • OperationImpl            |
| • ClassDeclarationImpl | • LinkedAttributeDeclaration | • OperationNotes           |
| • ClassInherits        | • LinkedClassBase            | • Parameter                |
| • ClassInterface       | • LinkedClassInterface       |                            |
| • ClassNotes           | • Namespace                  |                            |

### 5.1.2.2 Field Substitution Macros

The *field substitution* macros provide access to data in the model. In particular, they are used to access data fields from:

- Packages
- Classes
- Attributes
- Operations
- Parameters.

Field substitution macros are named according to Camel casing. By convention, the macro is prefixed with an abbreviated form of the corresponding model element. For example, attribute-related macros begin with **att**, as in the `%attName%` macro, to access the name of the attribute in scope.

The following table lists each of the field substitution macros with a description of the result.

#### Note:

Macros that represent checkboxes return a value of **T** if the box is selected. Otherwise the value is empty.

Macro Name	Description
attAlias	<b>Attributes</b> dialog: <b>Alias</b> .
attallowDuplicates	<b>Attributes Detail</b> dialog: <b>Allow Duplicates</b> checkbox.
attClassifierGUID	The unique GUID for the classifier of the current attribute.
attCollection	<b>Attributes Detail</b> dialog: <b>Attribute is a Collection</b> checkbox.
attConst	<b>Attributes</b> dialog: <b>Const</b> checkbox.
attContainerType	<b>Attributes Detail</b> dialog: <b>Container Type</b> .
attContainment	<b>Attributes</b> dialog: <b>Containment</b> .
attDerived	<b>Attributes</b> dialog: <b>Derived</b> checkbox.
attGUID	The unique GUID for the current attribute.
attInitial	<b>Attributes</b> dialog: <b>Initial</b> .
attIsEnumLiteral	<b>Attributes</b> dialog: <b>Is Literal</b> checkbox.
attLength	<b>Column</b> dialog: <b>Length</b> .
attLowerBound	<b>Attributes Detail</b> dialog: <b>Lower Bound</b> .
attName	<b>Attributes</b> dialog: <b>Name</b> .
attNotes	<b>Attributes</b> dialog: <b>Notes</b> .
attOrderedMultiplicity	<b>Attributes Detail</b> dialog: <b>Ordered Multiplicity</b> checkbox.
attProperty	<b>Attributes</b> dialog: <b>Property</b> checkbox.
attQualType	The attribute type qualified by the namespace path (if generating namespaces) and the classifier path (dot delimited). If the attribute classifier has not been set, is equivalent to the <code>attType</code> macro.
attScope	<b>Attributes</b> dialog: <b>Scope</b> .
attStatic	<b>Attributes</b> dialog: <b>Static</b> checkbox.



Macro Name	Description
attStereotype	<b>Attributes</b> dialog: <b>Stereotype</b> .
attType	<b>Attributes</b> dialog: <b>Type</b> .
attUpperBound	<b>Attributes Detail</b> dialog: <b>Upper Bound</b> .
attVolatile	<b>Attributes Detail</b> dialog: <b>Transient</b> checkbox.
classAbstract	<b>Class</b> dialog: <b>Abstract</b> checkbox.
classAlias	<b>Class</b> dialog: <b>Alias</b> .
classArguments	<b>Class Detail</b> dialog: <b>C++ Templates: Arguments</b> .
classAuthor	<b>Class</b> dialog: <b>Author</b> .
classBaseName	<b>Type Hierarchy</b> dialog: <b>Class Name</b> (for use where no connector exists between child and base Classes).
classBaseScope	The scope of the inheritance as reverse engineered. (For use where no connector exists between child and base Classes.)
classBaseVirtual	The virtual property of the inheritance as reverse engineered. (For use where no connector exists between child and base Classes.)
classComplexity	<b>Class</b> dialog: <b>Complexity</b> .
classCreated	The date and time the Class was created.
classGUID	The unique GUID for the current Class.
classHasConstructor	Looks at the list of methods in the current object and, depending on the conventions of the current language, returns <b>T</b> if one is a default constructor. Typically used with the <a href="#">genOptGenConstructor</a> <sup>94</sup> macro.
classHasCopyConstructor	Looks at the list of methods in the current object and, depending on the conventions of the current language, returns <b>T</b> if one is a copy constructor. Typically used with the <a href="#">genOptGenCopyConstructor</a> <sup>94</sup> macro.
classhasDestructor	Looks at the list of methods in the current object and, depending on the conventions of the current language, returns <b>T</b> if one is a destructor. Typically used with the <a href="#">genOptGenDestructor</a> <sup>94</sup> macro.
classHasParent	<b>True</b> , if the Class in scope has one or more base Classes.
classImports	<b>Code Gen</b> dialog: <b>Imports</b> .
classIsActive	<b>Class Advanced</b> dialog: <b>Is Active</b> checkbox.
classIsInstantiated	<b>True</b> , if the Class is an instantiated template Class.
classIsLeaf	<b>Class Advanced</b> dialog: <b>Is Leaf</b> checkbox.
classIsRoot	<b>Class Advanced</b> dialog: <b>Is Root</b> checkbox.
classIsSpecification	<b>Class Advanced</b> dialog: <b>Is Specification</b> checkbox.
classKeywords	<b>Class</b> dialog: <b>Keywords</b> .
classLanguage	<b>Class</b> dialog: <b>Language</b> .
classMacros	A space separated list of macros defined for the Class.

Macro Name	Description
classModified	The date and time the Class was last modified.
classMultiplicity	<b>Class Advanced</b> dialog: <b>Multiplicity</b> .
className	<b>Class</b> dialog: <b>Name</b> .
classNotes	<b>Class</b> dialog: <b>Note</b> .
classParamDefault	<b>Class Detail</b> dialog.
classParamName	<b>Class Detail</b> dialog.
classParamType	<b>Class Detail</b> dialog.
classPersistence	<b>Class</b> dialog: <b>Persistence</b> .
classPhase	<b>Class</b> dialog: <b>Phase</b> .
classQualName	The Class name prefixed by its outer Classes. Class names are separated by double colons (::).
classScope	<b>Class</b> dialog: <b>Scope</b> .
classStereotype	<b>Class</b> dialog: <b>Stereotype</b> .
classStatus	<b>Class</b> dialog: <b>Status</b> .
classVersion	<b>Class</b> dialog: <b>Version</b> .
connectorAlias	Connector <b>Properties</b> dialog: <b>Alias</b> .
connectorDestAccess	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Access</b> .
connectorDestAggregation	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Aggregation</b> .
connectorDestAlias	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Alias</b> .
connectorDestAllowDuplicates	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Allow Duplicates</b> checkbox.
connectorDestChangeable	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Changeable</b> .
connectorDestConstraint	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Constraint(s)</b> .
connectorDestContainment	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Containment</b> .
connectorDestDerived	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Derived</b> checkbox.
connectorDestDerivedUnion	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>DerivedUnion</b> checkbox.
connectorDestElem*	A set of macros that access a property of the element at the target end of a connector. The * (asterisk) is a wildcard that corresponds to any class substitution macro in this list; for example: <i>connectorDestElemAlias (classAlias)</i> , <i>connectorDestElemAuthor (classAuthor)</i> .
connectorDestElemType	The element type of the connector destination element. (Separate from the <i>connectorDestElem*</i> macros because there is no <i>classType</i> substitution macro.)
connectorDestMemberType	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Member Type</b> .
connectorDestMultiplicity	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Multiplicity</b> .

Macro Name	Description
connectorDestNavigability	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Navigability</b> .
connectorDestNotes	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Role Notes</b> .
connectorDestOrdered	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Ordered</b> checkbox.
connectorDestOwned	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Owned</b> checkbox.
connectorDestQualifier	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Qualifier(s)</b> .
connectorDestRole	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Role</b> .
connectorDestScope	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Target Scope</b> .
connectorDestStereotype	Connector <b>Properties</b> dialog, <b>Target Role</b> tab: <b>Stereotype</b> .
connectorDirection	Connector <b>Properties</b> : <b>Direction</b> .
connectorEffect	<b>Transition Constraints</b> dialog: <b>Effect</b> .
connectorGuard	<b>Object Flow</b> and <b>Transition Constraints</b> dialog: <b>Guard</b> .
connectorGUID	The unique GUID for the current connector.
connectorName	Connector <b>Properties</b> : <b>Name</b> .
connectorNotes	Connector <b>Properties</b> : <b>Notes</b> .
connectorSourceAccess	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Access</b> .
connectorSourceAggregation	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Aggregation</b> .
connectorSourceAlias	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Alias</b> .
connectorSourceAllowDuplicates	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Allow Duplicates</b> checkbox.
connectorSourceChangeable	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Changeable</b> .
connectorSourceConstraint	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Constraint(s)</b> .
connectorSourceContainment	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Containment</b> .
connectorSourceDerived	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Derived</b> checkbox.
connectorSourceDerivedUnion	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>DerivedUnion</b> checkbox.
connectorSourceElem*	A set of macros that access a property of the element at the source end of a connector. The * (asterisk) is a wildcard that corresponds to any class substitution macro in this list; for example: <i>connectorSourceElemAlias (classAlias)</i> , <i>connectorSourceElemAuthor (classAuthor)</i> .
connectorSourceElemType	The element type of the connector source element. (Separate from the <i>connectorSourceElem*</i> macros because there is no <i>classType</i> substitution macro.)
connectorSourceMemberType	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Member Type</b> .
connectorSourceMultiplicity	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Multiplicity</b> .
connectorSourceNavigability	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Navigability</b> .

Macro Name	Description
connectorSourceNotes	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Role Notes</b> .
connectorSourceOrdered	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Ordered</b> checkbox.
connectorSourceOwned	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Owned</b> checkbox.
connectorSourceQualifier	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Qualifier(s)</b> .
connectorSourceRole	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Role</b> .
connectorSourceScope	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Target Scope</b> .
connectorSourceStereotype	Connector <b>Properties</b> dialog, <b>Source Role</b> tab: <b>Stereotype</b> .
connectorStereotype	Connector <b>Properties</b> dialog: <b>Stereotype</b> .
connectorTrigger	<b>Transition Constraints</b> dialog: <b>Trigger</b> .
connectorType	The connector type; for example, Association or Generalization.
connectorWeight	<b>Object Flow Constraints</b> dialog: <b>Weight</b> .
constraintName	<b>Class</b> dialog, <b>Constraints</b> tab: <b>Name</b> .
constraintNotes	<b>Class</b> dialog, <b>Constraints</b> tab: <b>Notes</b> .
constraintStatus	<b>Class</b> dialog, <b>Constraints</b> tab: <b>Status</b> .
constraintType	<b>Class</b> dialog, <b>Constraints</b> tab: <b>Type</b> .
constraintWeight	<b>Class</b> dialog, <b>Constraints</b> tab: ordering (hand up/down) keys.
eaDateTime	The current time with format: <i>DD-MMM-YYYY HH:MM:SS AM/PM</i> .
eaGUID	A unique GUID for this generation.
eaVersion	Program Version (Located in an Enterprise Architect dialog by selecting <b>Help   About EA</b> ).
effortName	<b>Project Management</b> window: <b>Effort</b> .
effortNotes	<b>Project Management</b> window: <b>Notes</b> (unlabelled).
effortTime	<b>Project Management</b> window: <b>Time</b> .
effortType	<b>Project Management</b> window: <b>Type</b> .
elemType	The element type: Interface or Class.
fileExtension	The file type extension of the file being generated.
fileName	The name of the file being generated.
fileNameImpl	The filename of the implementation file for this generation, if applicable.
fileHeaders	<b>Code Gen</b> dialog: <b>Headers</b> .
fileImports	<b>Code Gen</b> dialog: <b>Imports</b> . For supported languages this also includes dependencies derived from associations.
filePath	The full path of the file being generated.
filePathImpl	The full path of the implementation file for this generation, if applicable.

Macro Name	Description
genOptActionScriptVersion	ActionScript Specifications dialog: <b>Default Version.</b>
genOptCDefaultAttributeType	C Specifications dialog: <b>Default Attribute Type.</b>
genOptCGenMethodNotesInBody	C Specifications dialog: <b>Method Notes In Implementation.</b>
genOptCGenMethodNotesInHeader	C Specifications dialog: <b>Method Notes In Header.</b>
genOptCSynchNotes	C Specifications dialog: <b>Synchronize Notes in Generation.</b>
genOptCSynchCFile	C Specifications dialog: <b>Synchronise Implementation file in Generation.</b>
genOptCDefaultSourceDirectory	C Specifications dialog: <b>Default Source Directory.</b>
genOptCNamespaceDelimiter	C Specifications dialog: <b>Namespace Delimiter.</b>
genOptCOperationRefParam	C Specifications dialog: <b>Reference as Operation Parameter.</b>
genOptCOperationRefParamStyle	C Specifications dialog: <b>Reference Parameter Style.</b>
genOptCOperationRefParamName	C Specifications dialog: <b>Reference Parameter Name.</b>
genOptCConstructorName	C Specifications dialog: <b>Default Constructor Name.</b>
genOptCDestructorName	C Specifications dialog: <b>Default Destructor Name.</b>
genOptCPPCommentStyle	C++ Specifications dialog: <b>Comment Style.</b>
genOptCPPDefaultAttributeType	C++ Specifications dialog: <b>Default Attribute Type.</b>
genOptCPPDefaultReferenceType	C++ Specifications dialog: <b>Default Reference Type.</b>
genOptCPPDefaultSourceDirectory	C++ Specifications dialog: <b>Default Source Directory.</b>
genOptCPPGenMethodNotesInHeader	C++ Specifications dialog: <b>Method Notes In Header</b> checkbox.
genOptCPPGenMethodNotesInBody	C++ Specifications dialog: <b>Method Notes In Body</b> checkbox.
genOptCPPGetPrefix	C++ Specifications dialog: <b>Get Prefix.</b>
genOptCPPHeaderExtension	C++ Specifications dialog: <b>Header Extension.</b>
genOptCPPSetPrefix	C++ Specifications dialog: <b>Set Prefix.</b>
genOptCPPSourceExtension	C++ Specifications dialog: <b>Source Extension.</b>
genOptCPPSynchCPPFile	C++ Specifications dialog: <b>Synchronize Notes.</b>
genOptCPPSynchNotes	C++ Specifications dialog: <b>Synchronize CPP File.</b>
genOptCSDDefaultAttributeType	C# Specifications dialog: <b>Default Attribute Type.</b>
genOptCSSourceExtension	C# Specifications dialog: <b>Default file extension.</b>
genOptCSGenDispose	C# Specifications dialog: <b>Generate Dispose.</b>
genOptCSGenFinalizer	C# Specifications dialog: <b>Generate Finalizer.</b>

Macro Name	Description
genOptCSGenNamespace	<b>C# Specifications</b> dialog: <b>Generate Namespace</b> .
genOptCSDefaultSourceDirectory	<b>C# Specifications</b> dialog: <b>Default Source Directory</b> .
genOptDefaultAssocAttName	<b>Attribute Specifications</b> dialog: <b>Default name for associated attrib.</b>
genOptDefaultConstructorScope	<b>Object Lifetimes</b> dialog: <b>Default Constructor Visibility</b> .
genOptDefaultCopyConstructorScope	<b>Object Lifetimes</b> dialog: <b>Default Copy Constructor Visibility</b> .
genOptDefaultDatabase	<b>Code Editors</b> dialog: <b>Default Database</b> .
genOptDefaultDestructorScope	<b>Object Lifetimes</b> dialog: <b>Default Destructor Constructor Visibility</b> .
genOptGenCapitalisedProperties	<b>Source Code Engineering</b> dialog: <b>Capitalize Attribute Names for Properties</b> checkbox.
genOptGenComments	<b>Source Code Engineering</b> dialog: <b>Generate Comments</b> checkbox.
genOptGenConstructor	<b>Object Lifetimes</b> dialog: <b>Generate Constructor</b> checkbox.
genOptGenConstructorInline	<b>Object Lifetimes</b> dialog: <b>Constructor Inline</b> checkbox.
genOptGenCopyConstructor	<b>Object Lifetimes</b> dialog: <b>Generate Copy Constructor</b> checkbox.
genOptGenCopyConstructorInline	<b>Object Lifetimes</b> dialog: <b>Copy Constructor Inline</b> checkbox.
genOptGenDestructor	<b>Object Lifetimes</b> dialog: <b>Generate Destructor</b> checkbox.
genOptGenDestructorInline	<b>Object Lifetimes</b> dialog: <b>Destructor Inline</b> checkbox.
genOptGenDestructorVirtual	<b>Object Lifetimes</b> dialog: <b>Virtual Destructor</b> checkbox.
genOptGenImplementedInterfaceOps	<b>Attribute/Operations Specifications</b> dialog: <b>Generate methods for implemented interfaces</b> checkbox.
genOptGenPrefixBoolProperties	<b>Source Code Engineering</b> dialog: <b>Use is prefix for boolean property Get()</b> .
genOptGenRoleNames	<b>Source Code Engineering</b> dialog: <b>Autogenerate role names when creating code</b> .
genOptGenUnspecAssocDir	<b>Source Code Engineering</b> dialog: <b>Do not generate members where Association direction is unspecified</b> checkbox.
genOptJavaDefaultAttributeType	<b>Java Specifications</b> dialog: <b>Default attribute type</b> .
genOptJavaGetPrefix	<b>Java Specifications</b> dialog: <b>Get Prefix</b> .
genOptJavaDefaultSourceDirectory	<b>Java Specifications</b> dialog: <b>Default Source Directory</b> .
genOptJavaSetPrefix	<b>Java Specifications</b> dialog: <b>Set Prefix</b> .
genOptJavaSourceExtension	<b>Java Specifications</b> dialog: <b>Source code extension</b> .
genOptPHPDefaultSourceDirectory	<b>PHP Specifications</b> dialog: <b>Default Source Directory</b> .
genOptPHPGetPrefix	<b>PHP Specifications</b> dialog: <b>Get Prefix</b> .
genOptPHPSetPrefix	<b>PHP Specifications</b> dialog: <b>Set Prefix</b> .

Macro Name	Description
genOptPHPSourceExtension	PHP Specifications dialog: <b>Default file extension.</b>
genOptPHPVersion	PHP Specifications dialog: <b>PHP Version.</b>
genOptPropertyPrefix	Source Code Engineering dialog: <b>Remove prefixes when generating Get/Set properties.</b>
genOptVBMultiUse	VB Specifications dialog: <b>Multiuse</b> checkbox.
genOptVBPersistable	VB Specifications dialog: <b>Persistable</b> checkbox.
genOptVBDataBindingBehavior	VB Specifications dialog: <b>Data binding behavior</b> checkbox.
genOptVBDataSourceBehavior	VB Specifications dialog: <b>Data source behavior</b> checkbox.
genOptVBGlobal	VB Specifications dialog: <b>Global namespace</b> checkbox.
genOptVBCreatable	VB Specifications dialog: <b>Creatable</b> checkbox.
genOptVBExposed	VB Specifications dialog: <b>Exposed</b> checkbox.
genOptVBMTS	VB Specifications dialog: <b>MTS Transaction Mode.</b>
genOptVBNetGenNamespace	VB.Net Specifications dialog: <b>Generate Namespace.</b>
genOptVBVersion	VB Specifications dialog: <b>Default Version.</b>
genOptWrapComment	Source Code Engineering dialog: <b>Wrap length for comment lines.</b>
importClassName	The name of the Class being imported.
importFileName	The filename of the Class being imported.
importFilePath	The full path of the Class being imported.
importFromAggregation	<b>T</b> if the Class has an Aggregation connector to a Class in this file, <b>F</b> otherwise.
importFromAssociation	<b>T</b> if the Class has an Association connector to a Class in this file, <b>F</b> otherwise.
importFromAtt	<b>T</b> if an attribute of a Class in the current file is of the type of this Class, <b>F</b> otherwise.
importFromDependency	<b>T</b> if the Class has a Dependency connector to a Class in this file, <b>F</b> otherwise.
importFromGeneralization	<b>T</b> if the Class has a Generalization connector to a Class in this file, <b>F</b> otherwise.
importFromMeth	<b>T</b> if a method return type of a Class in the current file is the type of this Class, <b>F</b> otherwise.
importFromParam	<b>T</b> if a method parameter of a Class in the current file is of the type of this Class, <b>F</b> otherwise.
importFromRealization	<b>T</b> if the Class has a Realization connector to a Class in this file, <b>F</b> otherwise.
importInFile	<b>T</b> if the Class is in the current file, <b>F</b> otherwise.
importPackagePath	The package path with a '.' separator of the Class being imported.

Macro Name	Description
ImportRelativeFilePath	The relative file path of the Class being imported from the file path of the file being generated.
linkAttAccess	Association Properties Target Role dialog: <b>Access</b> .
linkAttCollectionClass	The collection appropriate for the linked attribute in scope.
linkAttContainment	Association Properties Target Role dialog: <b>Containment</b> .
linkAttName	Association Properties dialog: <b>Target</b> .
linkAttNotes	Association Properties Target Role dialog: <b>Role Notes</b> .
linkAttQualName	The Association target qualified by the namespace path (if generating namespaces) and the classifier path (dot delimited).
linkAttRole	Association Properties Target Role dialog: <b>Role</b> .
linkAttStereotype	Association Properties Target Role dialog: <b>Stereotype</b> .
linkAttTargetScope	Association Properties Target Role dialog: <b>Target Scope</b> .
linkCard	Link Properties Target Role dialog: <b>Multiplicity</b> .
linkedFileLastWrite	Class Properties dialog: <b>Last Write</b> .
linkedFileNotes	Class Properties dialog: <b>Notes</b> .
linkedFilePath	Class Properties dialog: <b>File Path</b> .
linkedFileSize	Class Properties dialog: <b>Size</b> .
linkedFileType	Class Properties dialog: <b>Type</b> .
linkGUID	The unique GUID for the current connector.
linkParentName	Generalization Properties dialog: <b>Target</b> .
linkParentQualName	The Generalization target qualified by the namespace path (if generating namespaces) and the classifier path (dot delimited).
linkStereotype	The stereotype of the current connector.
linkVirtualInheritance	Generalization Properties dialog: <b>Virtual Inheritance</b> .
metricName	Project Management dialog, Metrics tab: <b>Metric</b> field.
metricNotes	Project Management dialog, Metrics tab: (Notes) field.
metricType	Project Management dialog, Metrics tab: <b>Type</b> field.
metricWeight	Project Management dialog, Metrics tab: <b>Weight</b> field.
opAbstract	Operation dialog: <b>Virtual</b> checkbox.
opAlias	Operation dialog: <b>Alias</b> .
opBehavior	Operation Behavior dialog: <b>Behavior</b> .
opCode	Operation Behavior dialog: <b>Initial Code</b> .
opConcurrency	Operation dialog: <b>Concurrency</b> .



Macro Name	Description
opConst	<b>Operation</b> dialog: <b>Const</b> checkbox.
opGUID	The unique GUID for the current operation.
opImplMacros	A space-separated list of macros defined in the implementation of this operation.
opIsQuery	<b>Operation</b> dialog: <b>IsQuery</b> checkbox.
opMacros	A space-separated list of macros defined in the declaration for this operation.
opName	<b>Operation</b> dialog: <b>Name</b> .
opNotes	<b>Operation</b> dialog: <b>Notes</b> .
opPure	<b>Operation</b> dialog: <b>Pure</b> checkbox.
opReturnArray	<b>Operation</b> dialog: <b>Return Array</b> checkbox.
opReturnClassifierGUID	The unique GUID for the classifier of the current operation.
opReturnQualType	The operation return type qualified by the namespace path (if generating namespaces) and the classifier path (dot delimited). If the return type classifier has not been set, is equivalent to the <i>opReturnType</i> macro.
opReturnType	<b>Operation</b> dialog: <b>Return Type</b> .
opScope	<b>Operation</b> dialog: <b>Scope</b> .
opStatic	<b>Operation</b> dialog: <b>Static</b> checkbox.
Operation dialog: Stereotype	<b>Operation</b> dialog: <b>Stereotype</b> .
opSynchronized	<b>Operation</b> dialog: <b>Synchronized</b> checkbox.
packageAbstract	<b>Package</b> dialog: <b>Abstract</b> .
packageAlias	<b>Package</b> dialog: <b>Alias</b> .
packageAuthor	<b>Package</b> dialog: <b>Author</b> .
packageComplexity	<b>Package</b> dialog: <b>Complexity</b> .
packageGUID	The unique GUID for the current package.
packageKeywords	<b>Package</b> dialog: <b>Keywords</b> .
packageLanguage	<b>Package</b> dialog: <b>Language</b> .
packageName	<b>Package</b> dialog: <b>Name</b> .
packagePath	The string representing the hierarchy of packages, for the Class in scope. Each package name is separated by a dot (.).
packagePhase	<b>Package</b> dialog: <b>Phase</b> .
packageScope	<b>Package</b> dialog: <b>Scope</b> .
packageStatus	<b>Package</b> dialog: <b>Status</b> .
packageStereotype	<b>Package</b> dialog: <b>Stereotype</b> .

Macro Name	Description
packageVersion	Package dialog: <b>Version</b> .
paramClassifierGUID	The unique GUID for the classifier of the current parameter.
paramDefault	Operation Parameters dialog: <b>Default</b> .
paramFixed	Operation Parameters dialog: <b>Fixed</b> checkbox.
paramGUID	The unique GUID for the current parameter.
paramsEnum	<b>True</b> , if the parameter uses the <i>enum</i> keyword (C++).
paramKind	Operation Parameters dialog: <b>Kind</b> .
paramName	Operation Parameters dialog: <b>Name</b> .
paramNotes	Operation Parameters dialog: <b>Notes</b> .
paramQualType	The parameter type qualified by the namespace path (if generating namespaces) and the classifier path (dot delimited). If the parameter classifier has not been set, is equivalent to the <i>paramType</i> macro.
paramType	Operation Parameters dialog: <b>Type</b> .
problemCompletedBy	Maintenance dialog, Element Issues tab: <b>Completed by</b> .
problemCompletedDate	Maintenance dialog, Element Issues tab: <b>Completed</b> .
problemHistory	Maintenance dialog, Element Issues tab: <b>History</b> .
problemName	Maintenance dialog, Element Issues tab: <b>Name</b> .
problemNotes	Maintenance dialog, Element Issues tab: <b>Description</b> .
problemPriority	Maintenance dialog, Element Issues tab: <b>Priority</b> .
problemRaisedBy	Maintenance dialog, Element Issues tab: <b>Raised by</b> .
problemRaisedDate	Maintenance dialog, Element Issues tab: <b>Raised</b> .
problemStatus	Maintenance dialog, Element Issues tab: <b>Status</b> .
problemVersion	Maintenance dialog, Element Issues tab: <b>Version</b> .
requirementDifficulty	Properties dialog: <b>Require</b> tab: <b>Difficulty</b> .
requirementLastUpdated	Properties dialog: <b>Require</b> tab: <b>Last Update</b> .
requirementName	Properties dialog: <b>Require</b> tab: <b>Short Description</b> .
requirementNotes	Properties dialog: <b>Require</b> tab: <b>Notes</b> .
requirementPriority	Properties dialog: <b>Require</b> tab: <b>Priority</b> .
requirementStatus	Properties dialog: <b>Require</b> tab: <b>Status</b> .
requirementType	Properties dialog: <b>Require</b> tab: <b>Type</b> .
resourceAllocatedTime	Project Management window, Resource Allocation tab: <b>Allocated Time</b> .
resourceEndDate	Project Management window, Resource Allocation tab: <b>End Date</b> .
resourceExpectedTime	Project Management window, Resource Allocation tab: <b>Expected Time</b> .

Macro Name	Description
resourceExpendedTime	Project Management window, Resource Allocation tab: <b>Time Expended</b> .
resourceHistory	Project Management window, Resource Allocation tab: <b>History</b> .
resourceName	Project Management window, Resource Allocation tab: <b>Resource</b> .
resourceNotes	Project Management window, Resource Allocation tab: <b>Description</b> .
resourcePercentCompleted	Project Management window, Resource Allocation tab: <b>Completed(%)</b> .
resourceRole	Project Management window, Resource Allocation tab: <b>Role</b> .
resourceStartDate	Project Management window, Resource Allocation tab: <b>Start Date</b> .
riskName	Project Management window, Risks tab: <b>Risk</b> .
riskNotes	Project Management window, Risks tab: (Notes).
riskType	Project Management window, Risks tab: <b>Type</b> .
riskWeight	Project Management window, Risks tab: <b>Weight</b> .
scenarioGUID	The unique ID for a scenario. Identifies the scenario unambiguously within a model.
scenarioName	Properties dialog, Scenario tab: <b>Scenario</b> .
scenarioNotes	Properties dialog, Scenario tab: (Notes).
scenarioType	Properties dialog, Scenario tab: <b>Type</b> .
testAcceptanceCriteria	Testing window: <b>Acceptance Criteria</b> .
testCheckedBy	Testing window: <b>Checked By</b> .
testDateRun	Testing window: <b>Last Run</b> .
testClass	The Testing window tab (the type of test defined): <b>Unit, Integration, System, Acceptance, Scenario</b> .
testInput	Testing window: <b>Input</b> .
testName	Testing window: <b>Test</b> .
testNotes	Testing window: <b>Description</b> .
testResults	Testing window: <b>Results</b> .
testRunBy	Testing window: <b>Run By</b> .
testStatus	Testing window: <b>Status</b> .
testType	Testing window: <b>Type</b> .

Field substitution macros can be used in one of two ways:

### Use 1: Direct Substitution

This form directly substitutes the corresponding value of the element in scope into the output.

Structure: %<macroName>%

Where `<macroName>` can be any of the macros listed above.

#### Examples:

- `%className%`
- `%opName%`
- `%attName%`

## Use 2: Conditional Substitution

This form of the macro enables alternative substitutions to be made depending on the macro's value.

Structure: `%<macroName> [ == "<test>" ] ? <subTrue> [ : <subFalse> ]%`

Where:

- `[ <text> ]` denotes that `<text>` is optional
- `<test>` is a string representing a possible value for the macro
- `<subTrue>` and `<subFalse>` can be a combination of quoted strings and the keyword value; where the value is used, it is replaced with the macro's value in the output.

#### Examples:

- `%classAbstract=="T" ? "pure" : ""%`
- `%opStereotype=="operator" ? "operator" : ""%`
- `%paramDefault != "" ? " = " value : ""%`

The above three examples output nothing if the condition fails. In this case the false condition can be omitted, resulting in the following usage:

#### Examples:

- `%classAbstract=="T" ? "pure" %`
- `%opStereotype=="operator" ? "operator" %`
- `%paramDefault != "" ? " = " value %`

The third example of both blocks shows a comparison checking for a non-empty value or existence. This test can also be omitted.

- `%paramDefault ? " = " value : ""%`
- `%paramDefault ? " = " value %`

All of the above examples containing `paramDefault` are equivalent. If the parameter in scope had a default value of **10**, the output from each of them would normally be:

= 10

#### Note:

In a conditional substitution macro, any white space following `<macroName>` is ignored. If white space is required in the output, it should be included within the quoted substitution strings.

### 5.1.2.3 Tagged Value Macros

*Tagged Value* macros are a special form of field substitution macros, which provide access to element tags and the corresponding Tagged Values.

## Use 1: Direct Substitution

This form of the macro directly substitutes the value of the named tag into the output.

Structure: `%<macroName>:"<tagName>"%`

`<macroName>` can be one of:

- `attTag`
- `classTag`
- `connectorDestTag`

- connectorSourceTag
- connectorTag
- linkAttTag
- linkTag
- opTag
- packageTag
- paramTag

This corresponds to the tags for attributes, Classes, operations, packages, parameters, connectors with both ends and connectors including the attribute end respectively.

`<tagName>` is a string representing the specific tag name.

**Examples:**

```
%opTag:"attribute"%
```

## Use 2: Conditional Substitution

This form of the macro mimics the conditional substitution defined for field substitution macros.

Structure: `%<macroName>:"<tagName>" [ == "<test>" ] ? <subTrue> [ : <subFalse> ]%`

Where:

- `<macroName>` and `<tagName>` are as defined above
- `[ <text> ]` denotes that `<text>` is optional
- `<test>` is a string representing a possible value for the macro
- `<subTrue>` and `<subFalse>` can be a combination of quoted strings and the keyword value. Where the value is used, it gets replaced with the macro's value in the output.

**Examples:**

```
%opTag:"opInline" ? "inline" : ""%
%opTag:"opInline" ? "inline"%
%classTag:"unsafe" == "true" ? "unsafe" : ""%
%classTag:"unsafe" == "true" ? "unsafe"%
```

Tagged Value macros use the same naming convention as field substitution macros.

### 5.1.2.4 Function Macros

*Function macros* are a convenient way of manipulating and formatting various element data. Each function macro returns a result string. There are two primary ways to use the results of function macros:

- Direct substitution of the returned string into the output, such as: `%TO_LOWER(attName)%`
- Storing the returned string as part of a variable definition such as: `$name = %TO_LOWER(attName)%`

Function macros can take parameters, which can be passed to the macros as:

- String literals, enclosed within double quotation marks
- Direct substitution macros without the enclosing percent signs
- Variable references
- Numeric literals.

Multiple parameters are passed using a comma-separated list.

The available function macros are described below. Parameters are denoted by angle brackets, as in: `FUNCTION_NAME(<param>)`.

**Note:**

Function macros are named according to the All-Caps style, as in: `%CONVERT_SCOPE(opScope)%`

#### **CONVERT\_SCOPE(<umlScope>)**

For use with supported languages. Converts `<umlScope>` to the appropriate scope keyword for the language being generated. The following table shows the conversion of `<umlScope>` with respect to the given language.

Language	Package	Public	Private	Protected
C++	public	public	private	protected
C#	internal	public	private	protected
Delphi	protected	public	private	protected
Java		public	private	protected
PHP	public	public	private	protected
VB	Protected	Public	Private	Protected
VB .Net	Friend	Public	Private	Protected

**COLLECTION\_CLASS(<language>)**

Gives the appropriate collection Class for the language specified for the current linked attribute.

**CSTYLE\_COMMENT(<wrap\_length>)**

Converts the notes for the element currently in scope to plain C-style comments, using /\* and \*/.

**DELPHI\_PROPERTIES(<scope>, <separator>, <indent>)**

Generates a Delphi property.

**DELPHI\_COMMENT(<wrap\_length>)**

Converts the notes for the element currently in scope to Delphi comments.

**EXEC\_ADD\_IN(<addin\_name>, <function\_name>, <prm\_1>, ..., <prm\_n>)**

Invokes an Enterprise Architect Add-In function, which can return a result string. <addin\_name> and <function\_name> specify the names of the Add-In and function to be invoked. Parameters to the Add-In function can be specified via parameters <prm\_1> to <prm\_n>. For example:

```
$result = %EXEC_ADD_IN("MyAddin","ProcessOperation",classGUID, opGUID)%
```

Any function that is to be called by the EXEC\_ADD\_IN macro must have two parameters: an *EA.Repository* object, and a *Variant* array that contains any additional parameters from the EXEC\_ADD\_IN call. Return type should be *Variant*. For example:

```
Public Function ProcessOperation(Repository As EA.Repository, args As Variant) As Variant
```

**FIND(<src>, <subString>)**

Position of the first instance of <subString> in <src>; -1 if none.

**GET\_ALIGNMENT()**

Returns a string where all of the text on the current line of output is converted into spaces and tabs.

**JAVADOC\_COMMENT(<wrap\_length>)**

Converts the notes for the element currently in scope to *javadoc*-style comments.

**LEFT(<src>, <count>)**

The first <count> characters of <src>.

**LENGTH(<src>)**

Length of <src>.

**MID(<src>, <count>)**

**MID(<src>, <start>, <count>)**

Substring of <src> starting at <start> and including <count> characters. Where <count> is omitted the rest of the string is included.

**PI(<option>, <value>, ...)**

Sets the PI for the current template to <value>. <option> controls when the new PI takes effect. Valid values are:

- *I, Immediate*: the new PI is generated before the next non-empty template line
- *N, Next*: the new PI is generated after the next non-empty template line.

Multiple pairs of options are allowed in one call. For more details, see the [description of PI](#).

**REMOVE\_DUPLICATES(<source>, <separator>)**

Where <source> is a <separator> separated list; this removes any duplicate or empty strings.

**REPLACE(<string>, <old>, <new>)**

Replaces all occurrences of <old> with <new> in the given string <string>.

**RESOLVE\_OP\_NAME()**

Resolves clashes in interface names where two method-from interfaces have the same name.

**RESOLVE\_QUALIFIED\_TYPE()****RESOLVE\_QUALIFIED\_TYPE(<separator>)****RESOLVE\_QUALIFIED\_TYPE(<separator>, <default>)**

Generates a qualified type for the current attribute, linked attribute, linked parent, operation, or parameter. Enables the specification of a separator other than . and a default value for when some value is required.

**RIGHT(<src>, <count>)**

The last <count> characters of <src>.

**TO\_LOWER(<string>)**

Converts <string> to lower case.

**TO\_UPPER(<string>)**

Converts <string> to upper case.

**TRIM(<string>)****TRIM(<string>, <trimChars>)**

Removes trailing and leading white spaces from <string>. If <trimChars> is specified, all leading and trailing characters in the set of <trimChars> are removed.

**TRIM\_LEFT(<string>)****TRIM\_LEFT(<string>, <trimChars>)**

Removes the specified leading characters from <string>.

**TRIM\_RIGHT(<string>)****TRIM\_RIGHT(<string>, <trimChars>)**

Removes the specified trailing characters from <string>.

**VB\_COMMENT(<wrap\_length>)**

Converts the notes for the element currently in scope to Visual Basic style comments.

**WRAP\_COMMENT(<comment>, <wrap\_length>, <indent>, <start\_string>)**

Wraps the text <comment> at width <wrap\_length> putting <indent> and <start\_string> at the beginning of each line. For example:

```
$behavior = %WRAP_COMMENT(opBehavior, "40", " ", "//")%
```

**Note:**

<wrap\_length> must still be passed as a string, even though **WRAP\_COMMENT** treats this parameter as an integer.

**WRAP\_LINES(<text>, <wrap\_length>, <start\_string>[, <end\_string>])**

Wraps <text> as designated to be <wrap\_length>, adding <start\_string> to the beginning of every line and <end\_string> to the end of the line if it is specified.

**XML\_COMMENT(<wrap\_length>)**

Converts the notes for the element currently in scope to XML-style comments.

**5.1.2.5 Control Macros**

*Control macros* are used to control the processing and formatting of the templates. The basic types of control macro include:

- The *list* macro, for generating multiple element features, such as attributes and operations
- The branching macros, which form *if-then-else* constructs to conditionally execute parts of a template
- The PI macro, which takes effect from the next non-empty line
- A PI [function macro](#)<sup>[10†]</sup> that enables setting PI to a variable and adds the ability to set the PI that is generated before the next line
- The PI macro for formatting new lines in the output
- The synchronization macros.

In general, control macros are named according to Camel casing.

**List**

The *list* macro is used to generate multiple elements. The basic structure is:

```
%list=<TemplateName> @separator=<string> @indent=<string> [<conditions>]%
```

where <string> is a double-quoted literal string and <TemplateName> can be one of the following template names:

- Attribute
- Class
- ClassBase
- ClassImpl
- ClassInterface
- Constraint
- Custom Template (custom templates enable you to define your own templates; for more information see [Custom Templates](#))<sup>[11†]</sup>.
- Effort
- InnerClass
- InnerClassImpl
- LinkedFile
- Metric
- Namespace
- Operation
- OperationImpl
- Parameter
- Problem



- Requirement
- Resource
- Risk
- Scenario
- Test

<conditions> is optional and appears the same as the conditions for *if* and *elseif* statements.

**Example:**

```
%list="Attribute" @separator="\n" @indent="  "%
```

The *separator* attribute, denoted above by *@separator*, specifies the space that should be used between the list items. This excludes the last item in the list.

The *indent* attribute, denoted by *@indent*, specifies the space by which each line in the generated output should be indented.

The above example would output the result of processing the *Attribute* template, for each attribute element of the Class in scope. The resultant list would separate its items with a single new line and indent them two spaces respectively. If the Class in scope had any stereotyped attributes, they would be generated using the appropriately specialized template.

There are some special cases to consider when using the *list* macro:

- If the *Attribute* template is used as an argument to the list macro, this also generates attributes derived from associations by executing the appropriate *LinkedAttribute* template
- If the *ClassBase* template is used as an argument to the list macro, this also generates Class bases derived from links in the model by executing the appropriate *LinkedClassBase* template
- If the *ClassInterface* template is used as an argument to the list macro, this also generates Class bases derived from links in the model by executing the appropriate *LinkedClassInterface* template
- If *InnerClass* or *InnerClassImpl* is used as an argument to the list macro, these Classes are generated using the *Class* and *ClassImpl* templates respectively. These arguments tell Enterprise Architect that it should process the templates based on the inner Classes of the Class in scope.

## Branching (if-then-else Constructs)

The CTF supports a limited form of branching through the following macros:

- *if*
- *elseif*
- *endif*
- *endTemplate*

The basic structure of the *if* and *elseif* macros is:

```
%if <test> <operator> <test>%
```

where <operator> can be one of:

- ==
- !=

and <test> can be one of:

- a string literal, enclosed within double quotation marks
- a direct substitution macro, without the enclosing percent signs
- a variable reference.

Branches can be nested, and multiple conditions can be specified using one of:

- and
- or.

**Note:**

When specifying multiple conditions, *and* and *or* have the same order of precedence, and conditions are processed left to right.

The *endif* or *endTemplate* macros must be used to signify the end of a branch. In addition, the *endTemplate* macro causes the template to return immediately, if the corresponding branch is being executed.

**Example:**

```
%if elemType == "Interface"%
;
%else%
%OperationBody%
%endif%
```

**Example:**

```
$bases=%list="ClassBase" @separator=", "%
$interfaces=%list="ClassInterface" @separator=", "%
%if $bases != "" and $interfaces != ""%
: $bases, $interfaces
%elseif $bases != ""%
: $bases
%elseif $interfaces != ""%
: $interfaces
%endif%
```

**The PI Macro**

There are two primary means of generating whitespace from the templates:

- Explicitly using the *newline*, *space* and *tab* characters (`\n`, `\t`) as part of Literal Text
- Using the *PI* macro to format lines in the template that result in non-empty substitutions in the output.

By default, each template line that generates a non-empty substitution also results in a newline being produced in the output. This behavior can be changed through the *PI* macro.

To demonstrate the use of the *PI* macro, consider the default *C# Operation* template:

```
%opTag:"Attribute"%
```

Default PI is `\n`, so any attributes would be on their own line

```
%PI=" "%
```

Blank lines have no effect on the output

```
%opTag:"unsafe"=="true" ? "unsafe" : ""%
```

Set the PI, so keywords are separated by a space

```
%CONVERT_SCOPE(opScope)%
```

Any keyword that does not apply - that is, the macro produces an empty result - does not result in a space

```
%opTag:"new"=="true" ? "new" : ""%
```

```
%opAbstract=="T" ? "abstract" : ""%
```

```
%opConst=="T" ? "sealed" : ""%
```

```
%opStatic=="T" ? "static" : ""%
```

```
%opTag:"extern"=="true" ? "extern" : ""%
```

```
%opTag:"delegate"=="true" ? "delegate" : ""%
```

```
%opTag:"override"=="true" ? "override" : ""%
```

```
%opTag:"virtual"=="true" ? "virtual" : ""%
```

Only one space is generated for this line

```
%opReturnType%%opReturnArray=="T" ? "[]" : ""%
```

```
%opStereotype=="operator" ? "operator" : ""%
```

```
%opName%%(list="Parameter" @separator=", "%)
```

The final line in the template does not generate a space

In the above example macros for the various keywords are to be arranged vertically for readability. In the output, however, each relevant keyword is to be separated by a single space. This is achieved by the line:

```
%PI=" "%
```

Notice how you do not specify the space between each of the possible keywords. This space is already implied by setting the PI to a single space. Essentially the PI acts as a convenience mechanism for formatting the output from within the templates.

The structure for setting the processing instruction is:

```
%PI=<value>%
```

where *<value>* can be a literal string enclosed by double quotes.

The following points apply to the *PI* macro:

- The value of the PI is not accessed explicitly
- Only template lines that result in a non-empty substitution cause the PI to be generated
- The last non-empty template line does not cause the PI to be generated
- The PI is not appended to the last substitution, regardless of which template line caused that substitution.

## Synchronization Macros

The *synchronization macros* are used to provide formatting hints to Enterprise Architect when inserting new sections into the source code, during forward synchronization. The values for synchronization macros must be set in the **File** templates.

The structure for setting synchronization macros is:

```
%<name>=<value>%
```

where *<name>* can be one of the macros listed below and *<value>* is a literal string enclosed by double quotes.

Macro Name	Description
<b>synchNewClassNotesSpace</b>	Space to append to a new Class note. Default value: \n.
<b>synchNewAttributeNotesSpace</b>	Space to append to a new attribute note. Default value: \n.
<b>synchNewOperationNotesSpace</b>	Space to append to a new operation note. Default value: \n.
<b>synchNewOperationBodySpace</b>	Space to append to a new operation body. Default value: \n.
<b>synchNamespaceBodyIndent</b>	Indent applied to Classes within non-global namespaces. Default value: \t.

### 5.1.2.6 EASL Code Generation Macros

Enterprise Architect provides two Enterprise Architect Simulation Library (EASL) code generation macros to generate code from behavioral models. These are:

- EASL\_GET and
- EASLList.

#### EASL\_GET

The *EASL\_GET* macro is used to retrieve a property or a collection of an EASL object. The EASL objects and the properties and collections for each object are identified in the [EASL Collections](#)<sup>[109]</sup> and [EASL Properties](#)<sup>[111]</sup> topics.

#### Syntax

```
$result = %EASL_GET(<<Property>>, <<Owner ID>>, <<Name>>)
```

where:

- <<Property>> is either "Property" or "Collection"
- <<OwnerID>> is the ID of the owner object for which the property/collection is to be retrieved
- <<Name>> is the name of the property or Collection being accessed
- \$result is the returned value; this is "" if not a valid property.

#### Example

```
$sPropName = %EASL_GET("Property", $context, "Name")%
```

#### EASLList

The *EASLList* macro is used to render each object in an EASL collection using the appropriate template.

#### Syntax

```
$result = %EASLList=<<TemplateName>> @separator=<<Separator>>
@indent=<<indent>> @owner=<<OwnedID>>
@collection=<<CollectionName>> @option1=<<OPTION1>>
@option2=<<OPTION2>>..... @optionN=<<OPTIONN>>%
```

where:

- <<TemplateName>> is the name of any [behavioral model template](#)<sup>[108]</sup> or [custom template](#)<sup>[117]</sup>
- <<Separator>> is a list separator (such as “\n”)
- <<indent>> is any indentation to be applied to the result
- <<owner>> is the ID of the object that contains the required collection
- <<CollectionName>> is the name of the required collection
- <<OPTION1>>...<<OPTION99>> are miscellaneous options that might be passed on the template; each option is given as an additional input parameter to the template
- \$result is the resultant value; this is “” if not a valid collection.

### Example

```
$sStates = %EASList="State" @separator="\n" @indent="\t"
           @owner=$StateMachineGUID @collection="States" @option=$sOption%
```

### Behavioral Model Templates

- Action
- Action Assignment
- Action Break
- Action Call
- Action Create
- Action Destroy
- Action If
- Action Loop
- Action Opaque
- Action Parallel
- Action RaiseEvent
- Action RaiseException
- Action Switch
- Behavior
- Behavior Body
- Behavior Declaration
- Behavior Parameter
- Call Argument
- Guard
- Property Object
- Property Declaration
- Property Notes
- State
- State CallBack
- State Enumerate
- State EnumeratedName
- StateMachine
- StateMachine HistoryVar
- Transition
- Transition Effect
- Trigger.

### 5.1.2.6.1 EASL Collections

This topic lists the EASL collections for each of the EASL objects, as retrieved by the [EASL\\_GET](#)<sup>[107]</sup> code generation macro.

#### Action

Collection Name	Description
Arguments	The Action's arguments.
SubActions	The sub-actions of the Action.

#### Behavior

Collection Name	Description
Actions	The Behavior's Actions.
Nodes	The Behavior's nodes.
Parameters	The Behavior's parameters.
Variables	The Behavior's variables.

#### Classifier

Collection Name	Description
AllStateMachines	All State Machines for the Classifier.
AsynchProperties	The asynchronous properties of the Classifier.
AsynchTriggers	The asynchronous triggers of the Classifier.
Behaviors	The behaviors of the Classifier.
Properties	The properties of the Classifier.
TimedProperties	The timed properties of the Classifier.
TimedTriggers	The timed triggers of the Classifier.

#### Construct

Collection Name	Description
AllChildren	The Construct's children.
ClientDependencies	The client dependencies on the Construct.
StereoTypes	The stereotypes of the Construct.
SupplierDependencies	The supplier dependencies on the Construct.

#### Node

Collection Name	Description
IncomingEdges	The Node's incoming edges.

Collection Name	Description
OutgoingEdges	The Node's outgoing edges.
SubNodes	The sub-nodes of the Node.

### State

Collection Name	Description
DoBehaviors	The State's Do behaviors.
EntryBehaviors	The State's Entry behaviors.
ExitBehaviors	The State's Exit behaviors.

### StateMachine

Collection Name	Description
AllFinalStates	The State Machine's final States.
AllStates	All States within the State Machine, including those within Submachine States.
DerivedTransitions	The State Machine's derived transitions with the associated valid effect.
States	The States within the State Machine.
Transitions	The transitions within the State Machine.
Vertices	The State Machine's vertices.

### Transition

Collection Name	Description
Effects	The Transition's effects.
Guards	The Transition's guards.
Triggers	The Transition's triggers.

### Trigger

Collection Name	Description
TriggeredTransitions	The triggered transitions associated with the Trigger.

### Vertex

Collection Name	Description
DerivedOutgoingTransitions	The Vertex's derived outgoing transitions after traversing the pseudo-nodes.
IncomingTransitions	The Vertex's incoming transitions.
OutgoingTransitions	The Vertex's outgoing transitions.

### 5.1.2.6.2 EASL Properties

This topic lists the EASL properties for each of the EASL objects, as retrieved by the [EASL\\_GET](#)<sup>[107]</sup> code generation macro.

#### Action

Property Name	Description
Behavior	The Action's associated behavior ( <i>Call Behavior</i> Action or <i>Call Operation</i> Action).
Body	The Action's body.
Context	The Action's context.
Guard	The Action's guard.
IsFinal	A check on whether the action is a final Action.
IsGuarded	A check on whether the action is a guarded Action.
IsInitial	A check on whether the action is an initial Action.
Kind	The Action's kind.
Next	The Action's next action.
Node	The Action's associated node in the graph.

#### Argument

Property Name	Description
Parameter	The ID of the Argument's associated parameter.
Value	The default value of the argument.

#### Behavior

Property Name	Description
InitialAction	The Behavior's initial action.
isReadOnly	The isReadOnly of the Behavior.
isSingleExecution	The isSingleExecution of the Behavior.
Kind	The kind of Behavior.
ReturnType	The return type of the Behavior.

#### CallEvent

Property Name	Description
Operation	The operation of the CallEvent.

### ChangeEvent

Property Name	Description
ChangeExpression	The change expression of the ChangeEvent.

### Classifier

Property Name	Description
HasBehaviors	A check on whether the Classifier has behavioral models (Activity and Interaction).
Language	The Classifier's language.
StateMachine	The State Machine of the Classifier.

### Condition

Property Name	Description
Expression	The Condition's expression.
Lower	The Condition's lower value.
Upper	The Condition's upper value.

### Construct

Property Name	Description
GetTaggedValue	The Property's Tagged Value.
IsStereotypeApplied	A check on whether a particular stereotype is applied to the Property.
Notes	Notes on the Property.
UMLType	The UML type of the Property.
Visibility	The visibility of the Property.

### Edge

Property Name	Description
From	The ID of the node from which the Edge arises.
To	The ID of the node at which the Edge is targeted.

### EventObject

Property Name	Description
EventKind	The event kind of the Event Object.



## Instance

Property Name	Description
Classifier	The classifier of the Instance.
Value	The value of the Instance.

## Parameter

Property Name	Description
Direction	The direction of the Parameter.
Type	The type of the Parameter.
Value	The value of the parameter.

## Primitive

Property Name	Description
FQName	The FQ name of the Primitive.
ID	The ID of the Primitive.
Name	The name of the Primitive.
ObjectType	The object type of the Primitive.
Parent	The IDParent of the Primitive.

## PropertyObject

Property Name	Description
BoundSize	The bound size of the PropertyObject (if it is a collection).
ClassifierStereoType	The stereotype of the PropertyObject's classifier.
IsAsynchProp	A check on whether the PropertyObject is an asynchronous property.
IsCollection	A check on whether the PropertyObject is a collection.
IsOrdered	A check on whether the PropertyObject is ordered (if it is a collection).
IsTimedProp	A check on whether the PropertyObject is a timed property.
Kind	The PropertyObject's kind.
LowerValue	The PropertyObject's lower value (if it is a collection).
Type	The PropertyObject's type.
UpperValue	The PropertyObject's upper value (if it is a collection).
Value	The PropertyObject's value.

### SignalEvent

Property Name	Description
Signal	The signal of the SignalEvent.

### State

Property Name	Description
HasSubMachine	A check on whether the State is a Submachine state.
IsFinalState	A check on whether the State is a final state.
SubMachine	Get the ID of the Submachine contained by the State (if applicable).

### StateMachine

Property Name	Description
HasSubMachineState	A check on whether the State Machine has a Submachine state.
InitialState	The State Machine's initial state.
SubMachineState	The State Machine's Submachine state.

### TimeEvent

Property Name	Description
When	The 'when' property of the TimeEvent.

### Transition

Property Name	Description
HasEffect	A check on whether the transition has a valid effect.
IsDerived	A check on whether the transition is a derived transition.
IsTranscend	A check on whether the transition transcends from one State Machine (Submachine state) to another.
IsTriggered	A check on whether the transition is triggered.
Source	The Transition's source.
Target	The Transition's target.

### Trigger

Property Name	Description
AsynchDestinationState	The asynchronous destination state of the Trigger (if it is an asynchronous trigger).
DependentProperty	The ID of the property associated with the Trigger.
Event	The Trigger's event.

Property Name	Description
Name	The Trigger's name.
Type	The Trigger's type.

## Vertex

Property Name	Description
IsHistory	A check on whether the vertex is a history state.
IsPseudoState	A check on whether the vertex is a pseudo state.
PseudoStateKind	The Vertex's pseudo-state kind.

### 5.1.3 Variables

Template variables provide a convenient way of storing and retrieving data within a template. This section explains how variables are [defined](#)<sup>[115]</sup> and [referenced](#)<sup>[116]</sup>.

#### Variable Definitions

Variable definitions take the basic form:

```
$<name> = <value>
```

where *<name>* can be any alpha-numeric sequence and *<value>* is derived from a macro or another variable.

A simple example definition would be:

```
$foo = %className%
```

Variables can be defined, using values from:

- Substitution, function or list macros
- String literals, enclosed within double quotation marks
- Variable references.

#### Definition Rules

The following rules apply to variable definitions:

- Variables have global scope within the template in which they are defined and are not accessible to other templates
- Each variable must be defined at the start of a line, without any intervening whitespace
- Variables are denoted by prefixing the name with \$, as in \$foo
- Variables do not have to be declared, prior to being defined
- Variables must be defined using either the assignment operator (=), or the addition-assignment operator (+=)
- Multiple terms can be combined in a single definition using the addition operator (+).

#### Examples

Using a substitution macro:

```
$foo = %opTag:"bar"%
```

Using a literal string:

```
$foo = "bar"
```

Using another variable:

```
$foo = $bar
```

Using a list macro:

```
$ops = %list="Operation" @separator="\n\n" @indent="\t"%
```

Using the addition-assignment operator (+=):

```
$body += %list="Operation" @separator="\n\n" @indent="\t"%
```

The above definition is equivalent to the following:

```
$body = $body + %list="Operation" @separator="\n\n" @indent="\t"%
```

Using multiple terms:

```
$templateArgs = %list="ClassParameter" @separator=", "%
$template = "template<" + $templateArgs + ">"
```

## Variable References

Variable values can be retrieved by using a reference of the form:

```
<name>
```

where <name> can be a previously defined variable.

Variable references can be used in one of the following ways:

- As part of a macro, such as the argument to a function macro
- As a term in a variable definition
- As a direct substitution of the variable value into the output.

### Note:

It is legal to reference a variable before it is defined. In this case, the variable is assumed to contain an empty string value: ""

### Example 1

Using variables as part of a macro. The following is an excerpt from the default C++ ClassNotes template.

```
$wrapLen = %genOptWrapComment%
$style = %genOptCPPCommentStyle%
```

Define variables to store the style and wrap length options.

```
%if $style == "XML.NET"%
%XML_COMMENT($wrapLen)%
%else%
%CSTYLE_COMMENT($wrapLen)%
%endif%
```

Reference to *\$style* as part of a condition.

Reference to *\$wrapLen* as an argument to function macro.

### Example 2

Using variable references as part of a variable definitions:

```
$foo = "foo"
$bar = "bar"
```

Define our variables.

```
$foobar = $foo + $bar
```

*\$foobar* now contains the value *foobar*.

### Example 3

Substituting variable values into the output

```
$bases=%classInherits%
```

Store the result of the *ClassInherits* template in *\$bases*.

```
Class %className%$bases
```

Now output the value of *\$bases* after the Class name.

## 5.2 The Code Template Editor in SDK

The **Code Template Editor** window is introduced in *Code Engineering Using UML Models*. The following topics describe how you use it to create custom templates:

- [Custom Templates](#)<sup>[117]</sup>
- [Override Default Templates](#)<sup>[118]</sup>
- [Add New Stereotyped Templates](#)<sup>[119]</sup>
- [Create Templates For Custom Languages](#)<sup>[120]</sup>

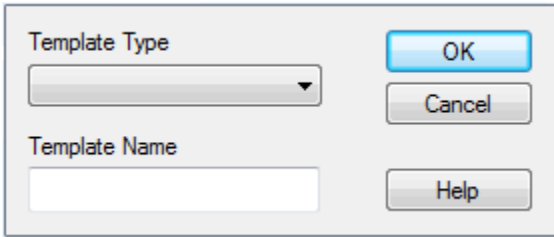
The Code Template Editor provides the facilities of the *Common Code Editor*, including Intellisense for the code generation template [macros](#)<sup>[87]</sup>. For more information on Intellisense and the Common Code Editor, see the *Code Editors* topic in *Using Enterprise Architect - UML Modeling Tool*.

### 5.2.1 Custom Templates

Custom templates enable you to generate an element in many different ways. Enterprise Architect enables you to define custom templates that are associated with given elements and call these templates from existing templates. You can even add stereotype overrides to your custom templates. For example, you might list all of your parameters and their notes in your method notes.

To create a new custom template, follow the steps below:

1. Select the **Settings | Code Generation Templates** menu option, or press **[Ctrl]+[Shift]+[P]**. The **Code Templates Editor** tab opens.
2. In the **Language** field, click on the drop-down arrow and select the appropriate language.
3. Click on the **Add New Custom Template** button. The **Create New Custom Template** dialog displays.

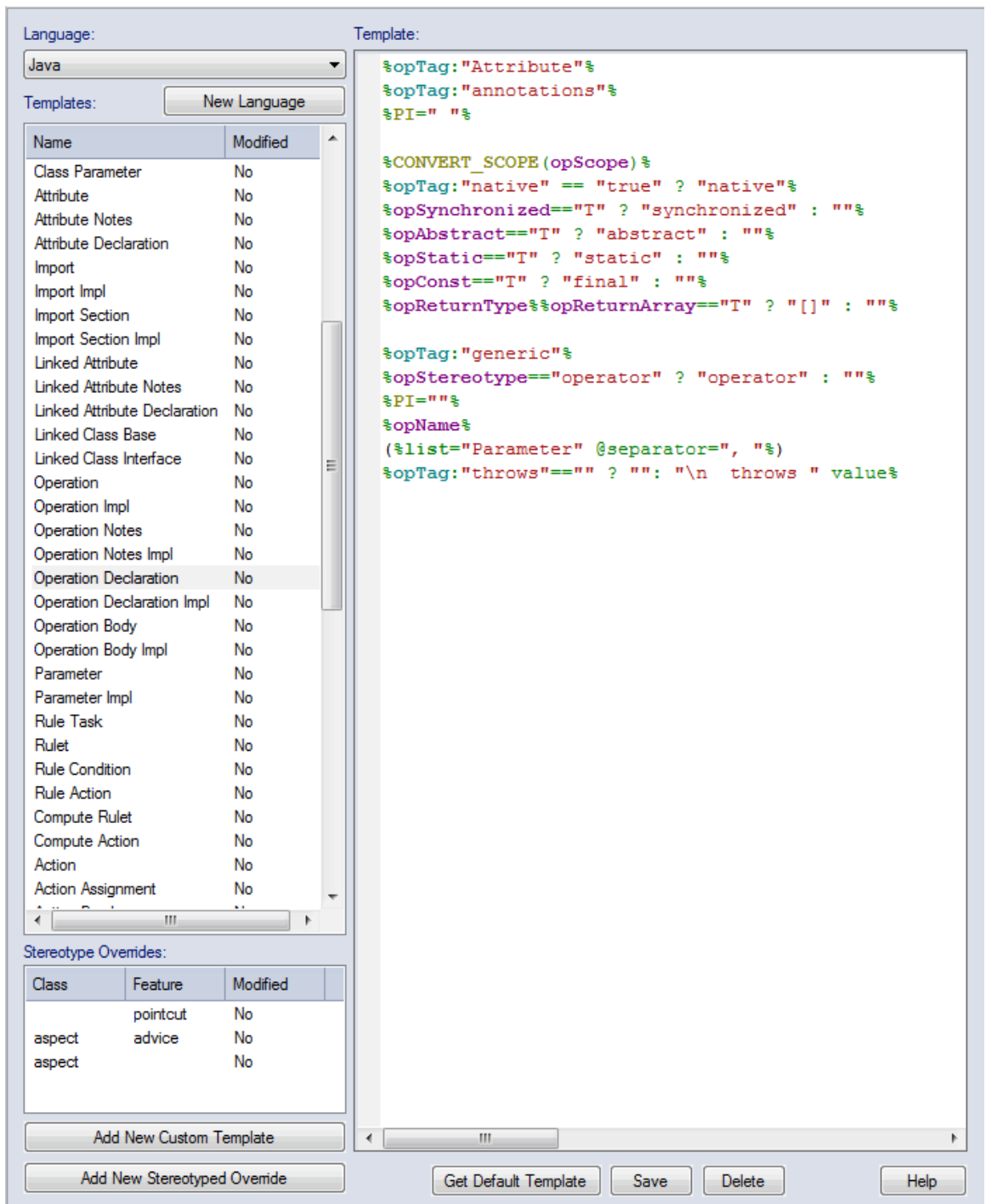


4. In the **Template Type** field, click on the drop-down arrow and select the appropriate element. The elements currently supported are:
  - Attribute
  - Class
  - Class Base
  - Class Interface
  - Class Parameter
  - Connector
  - Import
  - Linked Attribute
  - Linked Class Base
  - Linked Class Interface
  - Namespace
  - Operation
  - Parameter.

**Note:**

**<None>** requires special treatment. It enables the definition of a function macro that doesn't actually apply to any of the types, but must be called as a function to define variables *\$parameter1*, *\$parameter2* and so on for each value passed in.

5. In the **Template Name** field, type an appropriate name, then click on the **OK** button.
6. On the **Code Templates Editor** tab, the new template displays in the **Templates** list with the value **Yes** in the **Modified** field. The template is called **<Template Type>\_<Template Name>**.
7. Select the appropriate template from the **Templates** list and edit the contents in the **Template** field to meet your requirements.



- Click on the **Save** button. This stores the new stereotyped template in the .EAP file. The template is now available from the list of templates and via direct substitution for use.

### 5.2.2 Override Default Templates

Enterprise Architect has a set of built-in or default code generation templates. The **Code Templates Editor** enables you to modify these default templates, hence customizing the way in which Enterprise Architect generates code. You can choose to modify any or all of the base templates to achieve your required coding style.

Any templates that you have overridden are stored in the .EAP file. When generating code, Enterprise Architect first checks whether a template has been modified and if so, uses that template. Otherwise the

appropriate default template is used.

## Procedure

To override a default code generation template, follow the steps below.

1. Select the **Configuration | Code Generation Templates** menu option. The **Code Templates Editor** displays.
2. Select the appropriate language from the **Language** list.
3. Select one of the base templates from the **Templates** list.
4. If the base template has stereotyped overrides, you can select one of these from the **Stereotype Overrides** list.
5. In the **Code Templates Editor**, make the required modifications.
6. Click on the **Save** button. This stores the modified version of the template to the .EAP file. The template is marked as modified.

When generating code, Enterprise Architect now uses the overridden template, instead of the default template.

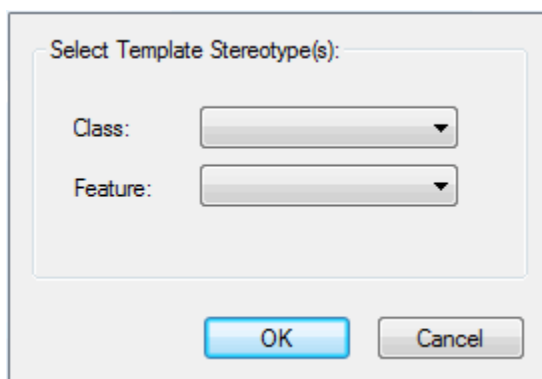
### 5.2.3 Add New Stereotyped Templates

Sometimes it is useful to define a specific code generation template for use with elements of a given stereotype. This enables different code to be generated for elements, depending on their stereotype. Enterprise Architect provides some default templates, which have been specialized for commonly used stereotypes in supported languages. For example the Operation Body template for C# has been specialized for the *property* stereotype, so that it automatically generates its constituent *get* and *set* methods. Users can override the default stereotyped templates as described in the previous topic. Additionally users can define templates for their own stereotypes, as described below.

#### Add a New Stereotyped Template

To override a default code generation template, follow the steps below.

1. Select the **Configuration | Code Generation Templates** menu option to open the **Code Templates Editor**.
2. Select the appropriate language, from the **Language** list.
3. Select one of the base templates, from the **Templates** list.
4. Click on the **Add New Stereotyped Override** button. The **New Template Override** dialog displays.



5. Select the required **Feature** and/or **Class** stereotype and click on the **OK** button.
6. The new stereotyped template override displays in **Stereotype Overrides** list, marked as modified.
7. Make the required modifications in the **Code Templates Editor**.
8. Click on the **Save** button. This stores the new stereotyped template in the .EAP file.

Enterprise Architect can now use the stereotyped template, when generating code for elements of that stereotype.

Note that Class and feature stereotypes can be combined to provide a further level of specialization for

features. For example, if properties should be generated differently when the Class has a stereotype *MyStereotype*, then both *property* and *MyStereotype* should be specified in the [New Template Override](#) dialog.

### 5.2.4 Create Custom Language Template

Enterprise Architect can forward generate code for languages that it does not specifically support, if the appropriate code generation templates are defined for that language. This topic outlines the steps required to define templates for custom languages.

#### Define a Template for a Custom Language

1. Create the custom language as a new product. To do this:
  - Select the **Settings | Code Datatypes** menu option. The [Programming Languages Datatypes](#) dialog displays.
  - In the **Product Name** field type the name of the new language, and in the **Datatype** field type a datatype (one is enough to declare that the new language exists). See the *Data Types* topic in *UML Model Management* for more details.
2. Select the **Settings | Code Generation Templates** menu option. The [Code Templates Editor](#) view displays.
3. In the **Language** field, click on the drop-down arrow and select the custom language.
4. From the [Templates](#) list, select one of the base templates.
5. Define the template using the [Code Templates Editor](#).
6. Click on the **Save** button. This stores the template in the .EAP file.
7. Repeat steps 1 to 6 for each of the relevant base templates for the custom language.

#### Note:

The *File* template must be defined for the custom language. The *File* template can then see the *Import Section*, *Namespace* and *Class* templates.



## 6 Enterprise Architect Add-In Model



### Introduction

Add-Ins enable you to add functionality to Enterprise Architect. The Enterprise Architect Add-In model builds on the features provided by the [Automation Interface](#) <sup>[176]</sup> to enable you to extend the Enterprise Architect user interface.

Add-Ins are ActiveX COM objects that expose public Dispatch methods. They have several advantages over stand-alone automation clients:

- Add-Ins can define Enterprise Architect menus and sub-menus
- Add-Ins receive notifications about various Enterprise Architect user-interface events including menu clicks and file changes
- Add-Ins can (and should) be written as in-process (DLL) components. This provides lower call overhead and better integration into the Enterprise Architect environment
- Because a current version of Enterprise Architect is already running there is no requirement to start a second copy of Enterprise Architect via the automation interface
- Because the Add-In receives object handles associated with the currently running copy of Enterprise Architect, more information is available about the current user's activity; for example, which diagram objects are selected
- You are not required to do anything other than to install the Add-In to make it usable; that is, you do not have to configure Add-Ins to run on your systems.

Because Enterprise Architect is constantly evolving in response to customer requests, the Add-In interface is flexible:

- The Add-In interface does not have its own version, rather it is identified by the version of Enterprise Architect it first appeared in; for example, the current version of the Enterprise Architect Add-In interface is version 2.1.
- When creating your Add-In, you do not have to subscribe to a type-library.

#### Note:

From Enterprise Architect release 7.0 Add-Ins created before 2004 are no longer supported. If an Add-In subscribes to the *Addn\_Tmpl.tlb* interface (2003 style), it will fail on load. In this event, contact the vendor or author of the Add-In and request an upgrade.

- Add-Ins do not have to implement methods that they never use.
- Add-Ins prompt users via context menus in the tree view and the diagram.
- Menu check and disable states can be controlled by the Add-In.

Add-Ins enhance the existing functionality of Enterprise Architect through a variety of mechanisms such as Scripts (see *Using Enterprise Architect - UML Modeling Tool*), [UML Profiles](#) <sup>[34]</sup> and the [Automation Interface](#) <sup>[176]</sup>. Once an Add-In is registered (see *Getting Started With Enterprise Architect*), it can be managed using the [Add-In Manager](#) <sup>[126]</sup>.

### Create and Use Add-Ins

This topic covers the following information on Add-Ins:

- [Add-In Tasks](#) <sup>[122]</sup>
- [Add-In Events](#) <sup>[127]</sup>
- [Broadcast Events](#) <sup>[133]</sup>
- [Custom Views](#) <sup>[164]</sup>
- [MDG Add-Ins](#) <sup>[165]</sup>

## 6.1 Add-In Tasks

This topic provides instructions on how to create, test, deploy and manage Add-Ins.

1. [Create an Add-In](#)<sup>[122]</sup>
  - [Define Menu Items](#)<sup>[122]</sup>
  - [Respond to Menu Events](#)<sup>[130]</sup>
  - [Handle Add-In Events](#)<sup>[127]</sup>
2. [Deploy your Add-In](#)<sup>[123]</sup>
  - [Potential Pitfalls](#)<sup>[124]</sup>
3. Manage Add-Ins
  - Register an Add-In (developed in-house or brought-in) - see the *Register Add-In* topic in *Getting Started with Enterprise Architect*
  - [The Add-In Manager](#)<sup>[126]</sup>

### 6.1.1 Create Add-Ins

Before you start you must have an application development tool that is capable of creating ActiveX COM objects supporting the IDispatch interface, such as:

- Borland Delphi
- Microsoft Visual Basic
- Microsoft Visual Studio .Net.

You should consider how to [define menu items](#)<sup>[122]</sup>. To help with this, you could review some [examples of Automation Interfaces](#) (this web page provides examples of code used to create Add-Ins for Enterprise Architect).

#### Create an Add-In

An Enterprise Architect Add-In can be created in four steps:

1. Use a development tool to create an ActiveX COM DLL project. Visual Basic users, for example, choose *File>Create New Project-ActiveX DLL*.
2. Connect to the interface using the syntax appropriate to the language as detailed in the [Connecting to the Interface](#)<sup>[176]</sup> topic.
3. Create a COM Class and implement each of the [general Add-In Events](#)<sup>[127]</sup> applicable to your Add-In. You only have to define methods for events to respond to.
4. Add a registry key that identifies your Add-In to Enterprise Architect, as described in the [Deploying Add-Ins](#)<sup>[123]</sup> topic.

#### 6.1.1.1 Define Menu Items

Menu items are defined by responding to the *GetMenuItems* event.

The first time this event is called, *MenuName* is an empty string, representing the top-level menu. For a simple Add-In with just a single menu option you can return a string; for example:

```
Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant
    EA_GetMenuItems = "&Joe's Add-In"
End Function
```

To define sub-menus, prefix a parent menu with a dash. Parent and sub-items are defined as follows:

```
Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant
    Select Case MenuName
        Case ""
            'Parent Menu Item
            EA_GetMenuItems = "-&Joe's Add-In"
        Case "-&Joe's Add-In"
            'Define Sub-Menu Items using the Array notation.
            'In this example, "Diagram" and "Treeview" compose the "Joe's Add-In" sub-menu.
            EA_GetMenuItems = Array("&Diagram", "&Treeview")
        Case Else
```

```

    MsgBox "Invalid Menu", vbCritical
End Select
End Function

```

Similarly, you can define further sub-items:

```

Function EA_GetMenuItems(Repository as EA.Repository, MenuLocation As String, MenuName As String) As Variant
    Select Case MenuName
    Case ""
        EA_GetMenuItems = "-Joe's Add-In"
    Case "-Joe's Add-In"
        EA_GetMenuItems = Array("-&Diagram", "&TreeView")
    Case "-&Diagram"
        EA_GetMenuItems = "&Properties"
    Case Else
        MsgBox "Invalid Menu", vbCritical
    End Select
End Function

```

To enable or disable menu options by default, you can use this method to show particular items to the user:

```

Sub EA_GetMenuState(Repository As EA.Repository, Location As String, MenuName As String, ItemName As String,
IsEnabled As Boolean, IsChecked As Boolean)
    Select Case Location
    Case "TreeView"
        'Always enable
    Case "Diagram"
        'Always enable
    Case "MainMenu"
        Select Case ItemName
        Case "&Translate", "Save &Project"
            If GetIsProjectSelected() Then
                IsEnabled = False
            End If
        End Select
    End Select
    IsChecked = GetIsCurrentSelection()
End Sub

```

### 6.1.1.2 Deploy Add-Ins

To deploy Add-Ins to users' sites, follow the steps below:

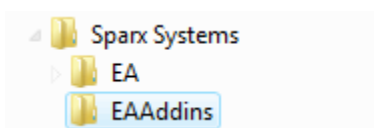
1. Add the Add-In DLL file to an appropriate directory on the user's computer; that is, C:\Program Files\[new dir].
2. Register the DLL as appropriate to your platform:
  - If compiled as a native Win32 DDL, such as VB6 or C++, register the DDL using the **regsvr32** command from the command prompt; for example:

```
regsvr32 "C:\Program Files\MyCompany\EAAAddin\EAAAddin.dll"
```

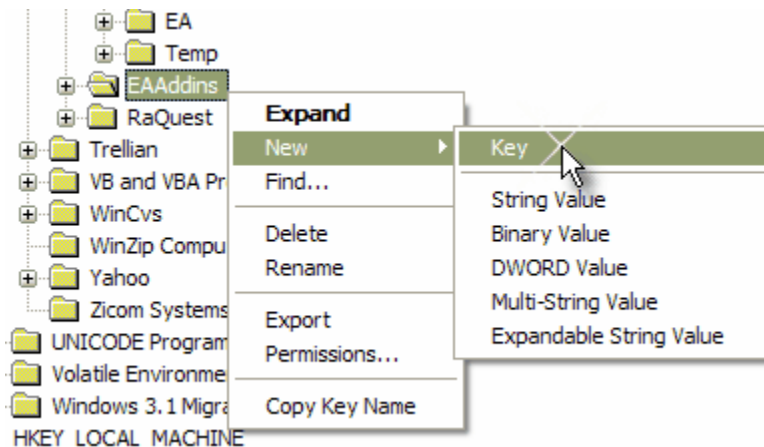
- If compiled as a .NET DLL, such as C# or VB.NET, register the DLL using the **RegAsm** command from the command prompt; for example:

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\RegAsm.exe "C:\Program Files\MyCompany\EAAAddin\EAAAddin.dll" /codebase
```

3. Place a new entry into the registry using the registry editor (run **regedit**) so that Enterprise Architect recognizes the presence of your Add-In.
4. Add a new key value **EAAAddins** under the location: HKEY\_CURRENT\_USER\Software\Sparx Systems

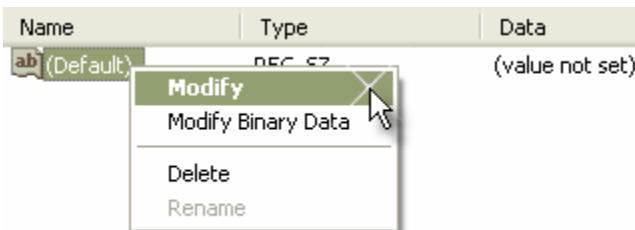


5. Add a new key under this key with the project name.

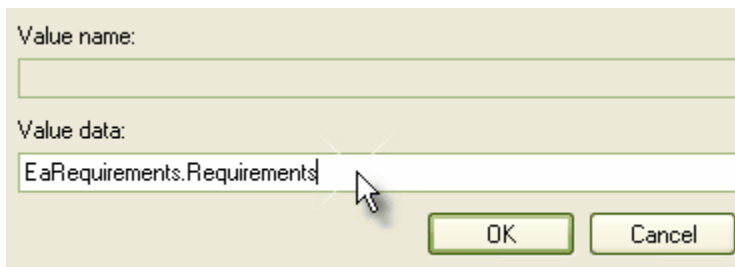
**Note:**

[ProjectName] is not necessarily the name of your DLL, but the name of the Project. In Visual Basic, this is the value for the property **Name** corresponding to the project file.

- Specify the default value by modifying the default value of the key.



- Enter the value of the key by typing in the [project name].[class name]; for example, EaRequirements.Requirements, where *EaRequirements* is the project name, as shown in the example below.



### 6.1.1.3 Tricks and Traps

#### Visual Basic 5/6 Users Note

Visual Basic 5/6 users should note that the version number of the Enterprise Architect interface is stored in the VBP project file in a form similar to the following:

```
Reference=*G{64FB2BF4-9EFA-11D2-8307-C45586000000}#2.2#0#..\..\..\Program Files\Sparx Systems\EA\EA.TLB#Enterprise Architect Object Model 2.02
```

If you experience problems moving from one version of Enterprise Architect to another, open the VBP file in a text editor and remove this line. Then open the project in Visual Basic and use *Project-References* to create a new reference to the Enterprise Architect Object model.

## Add-In Fails to Load

From Enterprise Architect release 7.0, Add-Ins created before 2004 are no longer supported. If an Add-In subscribes to the Addn\_Tmpl.tlb interface (2003 style), it will fail on load. In this event, contact the vendor or author of the Add-In and request an upgrade.

## Holding State Information

It is possible for an Add-In to hold state information, meaning that data can be stored in member variables in response to one event and retrieved in another. There are some dangers in doing this:

- Enterprise Architect Automation Objects do not update themselves in response to user activity, to activity on other workstations, or even to the actions of other objects in the same automation client. Retaining handles to such objects between calls can result in the second event querying objects that have no relationship with the current state of Enterprise Architect.
- When you close Enterprise Architect, all Add-Ins are asked to shut down. If there are any external automation clients Enterprise Architect must stay active, in which case all the Add-Ins are reloaded, losing all the data.
- Enterprise Architect acting as an automation client does not close if an Add-In still holds a reference to it (releasing all references in the Disconnect() event avoids this problem).

It is recommended that unless there is a specific reason for doing so, the Add-In should use the repository parameter and its method and properties to provide the necessary data.

## Enterprise Architect Not Closing

### .Net Specific Issues

Automation checks the use of objects and won't enable any of them to be destroyed until they are no longer being used.

As noted in the [Automation Interface](#)<sup>[179]</sup> topic, if your automation controller was written using the .NET framework, Enterprise Architect does not close even after you release all your references to it. To force the release of the COM pointers, call the memory management functions as shown below:

```
GC.Collect();
GC.WaitForPendingFinalizers();
```

Additionally, because automation clients hook into Enterprise Architect, which creates Add-Ins which in turn hook back into Enterprise Architect, it is possible to get into a deadlock situation where Enterprise Architect and the Add-Ins won't let go of one another and keep each other active. An Add-In might retain hooks into Enterprise Architect because:

- It keeps a private reference to an Enterprise Architect object (see [Holding State Information](#)<sup>[125]</sup> above), or
- It has been created by .NET and the GC mechanism hasn't got around to releasing it.

There are two actions required to avoid deadlock situations:

- Automation controllers must call Repository.CloseAddins() at some point (presumably at the end of processing).
- Add-Ins must release all references to Enterprise Architect in the Disconnect() event. See the [Add-In Methods](#)<sup>[127]</sup> topic for details.

It is possible that your Automation client controls a running instance of Enterprise Architect where the Add-Ins have not complied with the rule above. In this case you could call Repository.Exit() to terminate Enterprise Architect.

### Miscellaneous

In developing Add-Ins using the .Net framework you must select COM Interoperability in the project's properties in order for it to be recognized as an Add-In.

Some development environments do not automatically register COM DLLs on creation. You might have to do that manually before Enterprise Architect recognizes the Add-In.

You can use your private Add-In key (as required for Add-In deployment) to store configuration information pertinent to your Add-In.

## Concurrent Calls

In Enterprise Architect releases up to release 7.0, there is a possibility that Enterprise Architect could call two

Add-In methods concurrently if the Add-In calls:

- A message box
- A modal dialog
- VB DoEvents, .NET Application DoEvents or the equivalent in other languages.

In such cases, Enterprise Architect could initiate a second Add-In method before the first returns (re-entrancy). In release 7.0. and subsequent releases, Enterprise Architect cannot make such concurrent calls.

If developing Add-Ins, ensure that the Add-In users are running Enterprise Architect release 7.0 or a later release to avoid any risk of concurrent method calls.

## 6.2 The Add-In Manager

You can use the [Add-In Manager](#) to view what Add-Ins are available and to disable those not to be used.

Access the [Add-In Manager](#) dialog by selecting the **Add-Ins | Manage Add-Ins** menu option.

Available Add-Ins	Status	Load on Startup
DoDAF-MODAF	Enabled	<input checked="" type="checkbox"/>
Zachman Framework	Enabled	<input checked="" type="checkbox"/>

To enable an Add-In for use, select the **Load on Startup** check box. To disable an Add-In, deselect the checkbox.

### Note:

Enterprise Architect must be restarted for changes to take effect.

## 6.3 Add-In Search

Enterprise Architect enables Add-Ins to integrate with the [Model Search](#) (see *Using Enterprise Architect - UML Modeling Tool*). Searches can be defined that execute a method within your Add-In and display your results in an integrated way.

The method that runs the search must be structured in the following way:

*variant <method name> (Rep as Repository, SearchText as String, XMLResults as String)*

Parameter	Description
<b>Rep</b>	The currently open repository.
<b>SearchText</b>	An optional field that you can fill in through the <a href="#">Model Search</a> .
<b>XMLResults</b>	At completion of the method, this should contain the results for the search. The results should be an XML String that conforms to the <a href="#">Search Data Format</a> <sup>[127]</sup> .

### Return

The method must return a value for the results to be displayed.

## Advanced Usage

In addition to the displayed results, two additional hidden fields can be passed into the XML that provide special functionality.

### CLASSTYPE

Returning a field of CLASSTYPE, containing the Object\_Type value from the t\_object table, displays the appropriate icon in the column you place the field.

### CLASSGUID

Returning a field of CLASSGUID, containing an ea\_guid value, enables the **Model Search** to track the object in the **Project Browser** and open the **Properties** window for the element by double-clicking in the **Model Search**.

### 6.3.1 XML Format (Search Data)

The XML below provides the format for the *SearchData* parameter of the *RunModel* method. See the [Repository](#)<sup>[190]</sup> topic for more information.

```
<ReportViewData UID="MySearchID" >
  <!--
    //The UID attribute enables XML type searches to persist column information. That is, if you run the search, group by
    column or adjust column widths, then close the window and run the search again, the format/organization changes are
    retained.To avoid persisting column arrangements, leave the attribute value blank or remove it altogether.
    // Use this section to declare all possible fields - columns that appear in Enterprise Architect's search window - that
    are used below in <Rows/>.
    // The order of the columns of information to be appended here must match the order that the search run in
    Enterprise Architect would normally display.
    // Furthermore, if you append results onto a custom SQL Search, then the order used in your Custom SQL must
    match the order used below.
  -->

  <Fields>
    <Field name=""/>
    <Field name=""/>
    <Field name=""/>
    <Field name=""/>
  </Fields>

  <Rows>
    <Row>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
    </Row>
    <Row>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
    </Row>
    <Row>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
      <Field name="" value=""/>
    </Row>
  </Rows>
</ReportViewData>
```

## 6.4 Add-In Events

All Enterprise Architect Add-Ins can choose to respond to the following general Add-In events:

- [EA\\_Connect](#)<sup>[128]</sup>
- [EA\\_Disconnect](#)<sup>[128]</sup>
- [EA\\_GetMenuItems](#)<sup>[129]</sup>

- [EA\\_MenuClick](#)<sup>[130]</sup>
- [EA\\_GetMenuState](#)<sup>[129]</sup>
- [EA\\_ShowHelp](#)<sup>[132]</sup>
- [EA\\_OnOutputItemClicked](#)<sup>[131]</sup>
- [EA\\_OnOutputItemDoubleClicked](#)<sup>[131]</sup>

## 6.4.1 EA\_Connect

### Details

*EA\_Connect* events enable Add-Ins to identify their type and to respond to Enterprise Architect start up.

This event occurs when Enterprise Architect first loads your Add-In. Enterprise Architect itself is loading at this time so that while a Repository object is supplied, there is limited information that you can extract from it.

The chief uses for *EA\_Connect* are in initializing global Add-In data and for identifying the Add-In as an [MDG Add-In](#)<sup>[165]</sup>.

Also look at [EA\\_Disconnect](#)<sup>[128]</sup>.

### Syntax

**Function** *EA\_Connect(Repository As EA.Repository) As String*

The *EA\_Connect* function syntax has the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

String identifying a specialized type of Add-In:

Type	Details
"MDG"	MDG Add-Ins receive <a href="#">MDG Events</a> <sup>[165]</sup> and extra menu options.
""	None-specialized Add-In.

## 6.4.2 EA\_Disconnect

### Details

The *EA\_Disconnect* event enables the Add-In to respond to user requests to disconnect the model branch from an external project.

This function is called when the Enterprise Architect closes. If you have stored references to Enterprise Architect objects (not particularly recommended anyway), you must release them here.

In addition, .NET users must call memory management functions as shown below:

```
GC.Collect();
GC.WaitForPendingFinalizers();
```

Also look at [EA\\_Connect](#)<sup>[128]</sup>.

### Syntax

**Sub** *EA\_Disconnect()*



## Return Value

None.

### 6.4.3 EA\_GetMenuItems

#### Details

The *EA\_GetMenuItems* event enables the Add-In to provide the Enterprise Architect user interface with additional Add-In menu options in various context and main menus. When a user selects an Add-In menu option, an event is raised and passed back to the Add-In that originally defined that menu option.

This event is raised just before Enterprise Architect has to show particular menu options to the user, and its use is described in the [Define Menu Items](#)<sup>[122]</sup> topic.

Also look at:

- [EA\\_MenuClick](#)<sup>[130]</sup>
- [EA\\_GetMenuState](#)<sup>[129]</sup>.

#### Syntax

**Function** *EA\_GetMenuItems*(*Repository As EA.Repository, MenuLocation As String, MenuName As String*) *As Variant*

The *EA\_GetMenuItems* function syntax has the following elements:

Parameter	Type	Direction	Description
<b>MenuLocation</b>	<i>String</i>		String representing the part of the user interface that brought up the menu. Can be <i>TreeView</i> , <i>MainMenu</i> or <i>Diagram</i> .
<b>MenuName</b>	<i>String</i>		The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu it is an empty string.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

One of the following types:

- A string indicating the label for a single menu option.
- An array of strings indicating a multiple menu options.
- Empty (Visual Basic/VB.NET) or null (C#) to indicate that no menu should be displayed.

In the case of the top-level menu it should be a single string or an array containing only one item, or Empty/null.

### 6.4.4 EA\_GetMenuState

#### Details

The *EA\_GetMenuState* event enables the Add-In to set a particular menu option to either enabled or disabled. This is useful when dealing with locked packages and other situations where it is convenient to show a menu option, but not enable it for use.

This event is raised just before Enterprise Architect has to show particular menu options to the user. Its use is described in the [Define Menu Items](#)<sup>[122]</sup> topic.

Also look at [EA\\_GetMenuItems](#)<sup>[129]</sup>.

## Syntax

**Sub EA\_GetMenuState(Repository as EA.Repository, MenuLocation As String, MenuName as String, ItemName as String, IsEnabled as Boolean, IsChecked as Boolean)**

The *EA\_GetMenuState* function syntax has the following elements:

Parameter	Type	Direction	Description
<b>IsChecked</b>	<i>Boolean</i>		Boolean. Set to <b>True</b> to check this particular menu option.
<b>IsEnabled</b>	<i>Boolean</i>		Boolean. Set to <b>False</b> to disable this particular menu option.
<b>ItemName</b>	<i>String</i>		The name of the option actually clicked, for example, <i>Create a New Invoice</i> .
<b>MenuLocation</b>	<i>String</i>		String representing the part of the user interface that brought up the menu. Can be <i>TreeView</i> , <i>MainMenu</i> or <i>Diagram</i> .
<b>MenuName</b>	<i>String</i>		The name of the parent menu for which sub-items must be defined. In the case of the top-level menu it is an empty string.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

None.

## 6.4.5 EA\_MenuClick

### Details

*EA\_MenuClick* events are received by an Add-In in response to user selection of a menu option.

The event is raised when the user clicks on a particular menu option. When a user clicks on one of your non-parent menu options, your Add-In receives a *MenuClick* event, defined as follows:

```
Sub EA_MenuClick(Repository As EA.Repository, ByVal MenuName As String, ByVal ItemName As String)
```

The code below illustrates an example of use:

```
If MenuName = "-&Diagram" And ItemName = "&Properties" then
    MsgBox Repository.GetCurrentDiagram.Name, vbInformation
Else
    MsgBox "Not Implemented", vbCritical
End If
```

Notice that your code can directly access Enterprise Architect data and UI elements using [Repository](#)<sup>[189]</sup> methods.

Also look at [EA\\_GetMenuItems](#)<sup>[129]</sup>.

### Syntax

**Sub EA\_MenuClick(Repository As EA.Repository, MenuLocation As String, MenuName As String, ItemName As String)**

The *EA\_GetMenuClick* function syntax has the following elements:

Parameter	Type	Direction	Description
<b>ItemName</b>	<i>String</i>		The name of the option actually clicked, for example, <i>Create a</i>

Parameter	Type	Direction	Description
			<i>New Invoice.</i>
<b>MenuName</b>	<i>String</i>		The name of the parent menu for which sub-items are to be defined. In the case of the top-level menu it is an empty string.
<b>Repository</b>	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

None.

## 6.4.6 EA\_OnOutputItemClicked

### Details

*EA\_OnOutputItemClicked* events inform Add-Ins that the user has clicked on a list entry in the system tab or one of the user defined output tabs.

Usually an Add-In responds to this event in order to capture activity on an output tab they had previously created through a call to *Repository.AddTab()*.

Note that every loaded Add-In receives this event for every click on an output tab in Enterprise Architect - irrespective of whether the Add-In created that tab. Add-Ins should therefore check the **TabName** parameter supplied by this event to ensure that they are not responding to other Add-Ins' events.

Also look at [EA\\_OnOutputItemDoubleClicked](#)<sup>[131]</sup>.

### Syntax

**EA\_OnOutputItemClicked(Repository As EA.Repository, TabName As String, LineText As String, ID As Long)**

The *EA\_OnOutputItemClicked* function syntax has the following elements:

Parameter	Type	Direction	Description
<b>ID</b>	<i>Long</i>	IN	The ID value specified in the original call to <i>Repository.WriteOutput()</i> .
<b>LineText</b>	<i>String</i>	IN	The text that had been supplied as the String parameter in the original call to <i>Repository.WriteOutput()</i> .
<b>Repository</b>	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
<b>TabName</b>	<i>String</i>	IN	The name of the tab that the click occurred in. Usually this would have been created through <i>Repository.AddTab()</i> .

### Return Value

None.

## 6.4.7 EA\_OnOutputItemDoubleClicked

### Details

*EA\_OnOutputItemDoubleClicked* events informs Add-Ins that the user has used the mouse to double-click on

a list entry in one of the user-defined output tabs.

Usually an Add-In responds to this event in order to capture activity on an output tab they had previously created through a call to *Repository.AddTab()*.

Note that every loaded Add-In receives this event for every double-click on an output tab in Enterprise Architect - irrespective of whether the Add-In created that tab. Add-Ins should therefore check the **TabName** parameter supplied by this event to ensure that they are not responding to other Add-Ins' events.

Also look at [EA\\_OnOutputItemClicked](#)<sup>[137]</sup>.

## Syntax

**EA\_OnOutputItemDoubleClicked(Repository As EA.Repository, TabName As String, LineText As String, ID As Long)**

The *EA\_OnOutputItemClicked* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>ID</b>	<i>Long</i>	IN	The ID value specified in the original call to <i>Repository.WriteOutput()</i> .
<b>LineText</b>	<i>String</i>	IN	The text that had been supplied as the String parameter in the original call to <i>Repository.WriteOutput()</i> .
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
<b>TabName</b>	<i>String</i>	IN	The name of the tab that the click occurred in. Usually this would have been created through <i>Repository.AddTab()</i> .

## Return Value

None.

## 6.4.8 EA\_ShowHelp

### Details

The *EA\_ShowHelp* event enables the Add-In to show a help topic for a particular menu option. When the user has an Add-In menu option selected, pressing **[F1]** can be delegated to the required Help topic by the Add-In and a suitable help message shown.

This event is raised when the user presses **[F1]** on a menu option that is not a parent menu.

Also look at [EA\\_GetMenuItems](#)<sup>[129]</sup>.

### Syntax

**Sub EA\_ShowHelp(Repository as EA.Repository, MenuLocation As String, MenuName as String, ItemName as String)**

The *EA\_ShowHelp* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>ItemName</b>	<i>String</i>		The name of the option actually clicked, for example, <b>Create a New Invoice</b> .
<b>MenuLocation</b>	<i>String</i>		String representing the part of the user interface that brought up the menu. Can be <b>Treeview</b> , <b>MainMenu</b> or <b>Diagram</b> .
<b>MenuName</b>	<i>String</i>		The name of the parent menu for which sub-items are to be

Parameter	Type	Direction	Description
			defined. In the case of the top-level menu it is an empty string.
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

None.

## 6.5 Broadcast Events

### Overview

The following general Broadcast events are sent to all loaded Add-Ins. For an Add-In to receive the event, they must first implement the required automation event interface. If Enterprise Architect detects that the Add-In has the required interface, the event is dispatched on to the Add-In.

- [File Open Event](#) [133]
- [File Close Event](#) [134]
- [Pre-Deletion Events](#) [135]
- [Pre-New Events](#) [137]
- [Post-New Events](#) [142]
- [Technology Events](#) [146]
- [Context Item Events](#) [150]
- [Transformation Events](#) [146]
- [Compartment Events](#) [153]
- [Model Validation Broadcasts](#) [155]
- [Retrieve Model Template Event](#) [163]
- [Initialize Technology Event](#) [147]

[MDG Events](#) [165] add quite a number of additional events, but the Add-In must first have registered as an MDG-style Add-In, rather than as a generic Add-In.

### 6.5.1 EA\_FileOpen

#### Details

The *EA\_FileOpen* event enables the Add-In to respond to a *File Open* event. When Enterprise Architect opens a new model file, this event is raised and passed to all Add-Ins implementing this method.

The event occurs when the model being viewed by the Enterprise Architect user changes, for whatever reason (through user interaction or Add-In activity).

Also look at [EA\\_FileClose](#) [134] and [EA\\_FileNew](#) [134].

#### Syntax

**Sub EA\_FileOpen(Repository As EA.Repository)**

The *EA\_FileOpen* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model

Parameter	Type	Direction	Description
			data and user interface status information.

### Return Value

None.

## 6.5.2 EA\_FileClose

### Details

The *EA\_FileClose* event enables the Add-In to respond to a *File Close* event. When Enterprise Architect closes an opened Model file, this event is raised and passed to all Add-Ins implementing this method.

This event occurs when the model currently opened within Enterprise Architect is about to be closed (when another model is about to be opened or when Enterprise Architect is about to shutdown).

Also look at [EA\\_FileOpen](#)<sup>[133]</sup> and [EA\\_FileNew](#)<sup>[134]</sup>.

### Syntax

#### **Sub EA\_FileClose(Repository As EA.Repository)**

The *EA\_FileClose* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the Enterprise Architect model about to be closed. Poll its members to retrieve model data and user interface status information.

### Return Value

None.

## 6.5.3 EA\_FileNew

### Details

The *EA\_FileNew* event enables the Add-In to respond to a *File New* event. When Enterprise Architect creates a new model file, this event is raised and passed to all Add-Ins implementing this method.

The event occurs when the model being viewed by the Enterprise Architect user changes, for whatever reason (through user interaction or Add-In activity).

Also look at [EA\\_FileClose](#)<sup>[134]</sup> and [EA\\_FileOpen](#)<sup>[133]</sup>.

### Syntax

#### **Sub EA\_FileNew(Repository As EA.Repository)**

The *EA\_FileNew* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

None.

### 6.5.4 Pre-Deletion Events

Enterprise Architect Add-Ins can respond to requests to delete elements, connectors, diagrams, packages and technologies using the following broadcast events:

- [EA\\_OnPreDeleteElement](#)<sup>[135]</sup>
- [EA\\_OnPreDeleteConnector](#)<sup>[135]</sup>
- [EA\\_OnPreDeleteDiagram](#)<sup>[136]</sup>
- [EA\\_OnPreDeletePackage](#)<sup>[136]</sup>.

#### 6.5.4.1 EA\_OnPreDeleteElement

##### Details

*EA\_OnPreDeleteElement* notifies Add-Ins that an element is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the element.

This event occurs when a user deletes an element from the **Project Browser** or on a diagram. The notification is provided immediately before the element is deleted, so that the Add-In can disable deletion of the element.

##### Syntax

**Function** *EA\_OnPreDeleteElement(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPreDeleteElement* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the element to be deleted: <ul style="list-style-type: none"> <li>• <i>ElementID</i>: A long value corresponding to <i>Element.ElementID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** to enable deletion of the element from the model. Return **False** to disable deletion of the element.

#### 6.5.4.2 EA\_OnPreDeleteConnector

##### Details

*EA\_OnPreDeleteConnector* notifies Add-Ins that a connector is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the connector.

This event occurs when a user attempts to permanently delete a connector on a diagram. The notification is provided immediately before the connector is deleted, so that the Add-In can disable deletion of the connector.

##### Syntax

**Function** *EA\_OnPreDeleteConnector(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPreDeleteConnector* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <small>[207]</small>	IN	Contains the following <i>EventProperty Objects</i> for the connector to be deleted: <ul style="list-style-type: none"> <li><i>ConnectorID</i>: A long value corresponding to <i>Connector.ConnectorID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to enable deletion of the connector from the model. Return **False** to disable deletion of the connector.

#### 6.5.4.3 EA\_OnPreDeleteDiagram

##### Details

*EA\_OnPreDeleteDiagram* notifies Add-Ins that a diagram is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the diagram.

This event occurs when a user attempts to permanently delete a diagram from the **Project Browser**. The notification is provided immediately before the diagram is deleted, so that the Add-In can disable deletion of the diagram.

##### Syntax

**Function** *EA\_OnPreDeleteDiagram(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPreDeleteDiagram* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <small>[207]</small>	IN	Contains the following <i>EventProperty Objects</i> for the connector to be deleted: <ul style="list-style-type: none"> <li><i>DiagramID</i>: A long value corresponding to <i>Diagram.DiagramID</i></li> </ul>
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to enable deletion of the diagram from the model. Return **False** to disable deletion of the diagram.

#### 6.5.4.4 EA\_OnPreDeletePackage

##### Details

*EA\_OnPreDeletePackage* notifies Add-Ins that a package is to be deleted from the model. It enables Add-Ins to permit or deny deletion of the package.

This event occurs when a user attempts to permanently delete a package from the **Project Browser**. The notification is provided immediately before the package is deleted, so that the Add-In can disable deletion of the package.



## Syntax

**Function** `EA_OnPreDeletePackage(Repository As EA.Repository, Info As EA.EventProperties) As Boolean`

The `EA_OnPreDeletePackage` function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the connector to be deleted: <ul style="list-style-type: none"> <li><i>PackageID</i>: A long value corresponding to <i>Package.PackageID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** to enable deletion of the package from the model. Return **False** to disable deletion of the package.

### 6.5.5 Pre-New Events

Enterprise Architect Add-Ins can respond to requests to create new elements, connectors, objects, attributes, methods and packages using the following broadcast events:

- [EA\\_OnPreNewElement](#)<sup>[137]</sup>
- [EA\\_OnPreNewConnector](#)<sup>[138]</sup>
- [EA\\_OnPreNewDiagramObject](#)<sup>[139]</sup>
- [EA\\_OnPreNewAttribute](#)<sup>[139]</sup>
- [EA\\_OnPreNewMethod](#)<sup>[140]</sup>
- [EA\\_OnPreNewPackage](#)<sup>[141]</sup>.

#### 6.5.5.1 EA\_OnPreNewElement

### Details

`EA_OnPreNewElement` notifies Add-Ins that a new element is about to be created on a diagram. It enables Add-Ins to permit or deny creation of the new element.

This event occurs when a user drags a new element from the Enterprise Architect UML **Toolbox** or **Resources** window onto a diagram. The notification is provided immediately before the element is created, so that the Add-In can disable addition of the element.

Also look at [EA\\_OnPostNewElement](#)<sup>[142]</sup>.

## Syntax

**Function** `EA_OnPreNewElement(Repository As EA.Repository, Info As EA.EventProperties) As Boolean`

The `EA_OnPreNewElement` function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the element to be created: <ul style="list-style-type: none"> <li><i>Type</i>: A string value corresponding to <i>Element.Type</i></li> </ul>

Parameter	Type	Direction	Description
			<ul style="list-style-type: none"> <li><i>Stereotype</i>: A string value corresponding to <i>Element.Stereotype</i></li> <li><i>ParentID</i>: A long value corresponding to <i>Element.ParentID</i></li> <li><i>DiagramID</i>: A long value corresponding to the ID of the diagram to which the element is being added.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to enable addition of the new element to the model. Return **False** to disable addition of the new element.

### 6.5.5.2 EA\_OnPreNewConnector

#### Details

*EA\_OnPreNewConnector* notifies Add-Ins that a new connector is about to be created on a diagram. It enables Add-Ins to permit or deny creation of a new connector.

This event occurs when a user drags a new connector from the Enterprise Architect UML **Toolbox** or **Resources** window, onto a diagram. The notification is provided immediately before the connector is created, so that the Add-In can disable addition of the connector.

Also look at [EA\\_OnPostNewConnector](#)<sup>[138]</sup>.

#### Syntax

**Function** *EA\_OnPreNewConnector*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPreNewConnector* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the connector to be created: <ul style="list-style-type: none"> <li><i>Type</i>: A string value corresponding to <i>Connector.Type</i></li> <li><i>Subtype</i>: A string value corresponding to <i>Connector.Subtype</i></li> <li><i>Stereotype</i>: A string value corresponding to <i>Connector.Stereotype</i></li> <li><i>ClientID</i>: A long value corresponding to <i>Connector.ClientID</i></li> <li><i>SupplierID</i>: A long value corresponding to <i>Connector.SupplierID</i></li> <li><i>DiagramID</i>: A long value corresponding to <i>Connector.DiagramID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to enable addition of the new connector to the model. Return **False** to disable addition of the new connector.

### 6.5.5.3 EA\_OnPreNewDiagramObject

#### Details

*EA\_OnPreNewDiagramObject* notifies Add-Ins that a new diagram object is about to be dropped on a diagram. It enables Add-Ins to permit or deny creation of the new object.

This event occurs when a user drags an object from the Enterprise Architect **Project Browser** or **Resources** window onto a diagram. The notification is provided immediately before the object is created, so that the Add-In can disable addition of the object.

Also look at [EA\\_OnPostNewDiagramObject](#)<sup>[143]</sup>.

#### Syntax

**Function** *EA\_OnPreNewDiagramObject*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPreNewDiagramObject* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the object to be created: <ul style="list-style-type: none"> <li><i>Type</i>: A string value corresponding to <i>Object.Type</i></li> <li><i>Stereotype</i>: A string value corresponding to <i>Object.Stereotype</i></li> <li><i>ParentID</i>: A long value corresponding to <i>Object.ParentID</i></li> <li><i>DiagramID</i>: A long value corresponding to the ID of the diagram to which the object is being added.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Return **True** to enable addition of the object to the model. Return **False** to disable addition of the object.

### 6.5.5.4 EA\_OnPreNewAttribute

#### Details

*EA\_OnPreNewAttribute* notifies Add-Ins that a new attribute is about to be created on an element. It enables Add-Ins to permit or deny creation of the new attribute.

This event occurs when a user creates a new attribute on an element by either drag-dropping from the **Project Browser**, using the **Attributes Properties** dialog, or using the in-place editor on the diagram. The notification is provided immediately before the attribute is created, so that the Add-In can disable addition of the attribute.

Also look at [EA\\_OnPostNewAttribute](#)<sup>[144]</sup>.

#### Syntax

**Function** *EA\_OnPreNewAttribute*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPreNewAttribute* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a>	IN	Contains the following <i>EventProperty Objects</i> for the attribute to be created:

Parameter	Type	Direction	Description
	<a href="#">EA.Attribute</a> <sup>[207]</sup>		<ul style="list-style-type: none"> <li><i>Type</i>: A string value corresponding to <i>Attribute.Type</i></li> <li><i>Stereotype</i>: A string value corresponding to <i>Attribute.Stereotype</i></li> <li><i>ParentID</i>: A long value corresponding to <i>Attribute.ParentID</i></li> <li><i>ClassifierID</i>: A long value corresponding to <i>Attribute.ClassifierID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to enable addition of the new attribute to the model. Return **False** to disable addition of the new attribute.

#### 6.5.5.5 EA\_OnPreNewMethod

### Details

*EA\_OnPreNewMethod* notifies Add-Ins that a new method is about to be created on an element. It enables Add-Ins to permit or deny creation of the new method.

This event occurs when a user creates a new method on an element by either drag-dropping from the **Project Browser**, using the method **Properties** dialog, or using the in-place editor on the diagram. The notification is provided immediately before the method is created, so that the Add-In can disable addition of the method.

Also look at [EA\\_OnPostNewMethod](#)<sup>[144]</sup>.

### Syntax

**Function** *EA\_OnPreNewMethod*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPreNewMethod* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the method to be created: <ul style="list-style-type: none"> <li><i>ReturnType</i>: A string value corresponding to <i>Method.ReturnType</i></li> <li><i>Stereotype</i>: A string value corresponding to <i>Method.Stereotype</i></li> <li><i>ParentID</i>: A long value corresponding to <i>Method.ParentID</i></li> <li><i>ClassifierID</i>: A long value corresponding to <i>Method.ClassifierID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to enable addition of the new method to the model. Return **False** to disable addition of the new method.

### 6.5.5.6 EA\_OnPreNewPackage

#### Details

*EA\_OnPreNewPackage* notifies Add-Ins that a new package is about to be created in the model. It enables Add-Ins to permit or deny creation of the new package.

This event occurs when a user drags a new package from the Enterprise Architect UML **Toolbox** or **Resources** window onto a diagram, or by selecting the **New Package** icon from the **Project Browser**. The notification is provided immediately before the package is created, so that the Add-In can disable addition of the package.

Also look at [EA\\_OnPostNewPackage](#)<sup>[145]</sup>.

#### Syntax

**Function** *EA\_OnPreNewPackage*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPreNewPackage* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the package to be created: <ul style="list-style-type: none"> <li><i>Stereotype</i>: A string value corresponding to <i>Package.Stereotype</i></li> <li><i>ParentID</i>: A long value corresponding to <i>Package.ParentID</i></li> <li><i>DiagramID</i>: A long value corresponding to the ID of the diagram to which the package is being added.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Return **True** to enable addition of the new package to the model. Return **False** to disable addition of the new package.

### 6.5.6 EA\_OnPreExitInstance

#### Details

*EA\_OnPreExitInstance* is not currently used.

#### Syntax

**Sub** *EA\_OnPreExitInstance*(*Repository As EA.Repository*)

The *EA\_OnPreExitInstance* function syntax contains the following element:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

None.

## 6.5.7 Post-New Events

Enterprise Architect Add-Ins can respond to the creation of new elements, connectors, objects, attributes, methods and packages using the following broadcast events:

- [EA\\_OnPostNewElement](#)<sup>[142]</sup>
- [EA\\_OnPostNewConnector](#)<sup>[142]</sup>
- [EA\\_OnPostNewDiagramObject](#)<sup>[143]</sup>
- [EA\\_OnPostNewAttribute](#)<sup>[144]</sup>
- [EA\\_OnPostNewMethod](#)<sup>[144]</sup>
- [EA\\_OnPostNewPackage](#)<sup>[145]</sup>

### 6.5.7.1 EA\_OnPostNewElement

#### Details

*EA\_OnPostNewElement* notifies Add-Ins that a new element has been created on a diagram. It enables Add-Ins to modify the element upon creation.

This event occurs after a user has dragged a new element from the Enterprise Architect UML **Toolbox** or **Resources** window onto a diagram. The notification is provided immediately after the element is added to the model. Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA\\_OnPreNewElement](#)<sup>[137]</sup>.

#### Syntax

**Function** *EA\_OnPostNewElement(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPostNewElement* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>Info</b>	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the new element: <ul style="list-style-type: none"> <li>• <i>ElementID</i>: A long value corresponding to <i>Element.ElementID</i>.</li> </ul>
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Return **True** if the element has been updated during this notification. Return **False** otherwise.

### 6.5.7.2 EA\_OnPostNewConnector

#### Details

*EA\_OnPostNewConnector* notifies Add-Ins that a new connector has been created on a diagram. It enables Add-Ins to modify the connector upon creation.

This event occurs after a user has dragged a new connector from the Enterprise Architect UML **Toolbox** or **Resources** window onto a diagram. The notification is provided immediately after the connector is added to the model. Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA\\_OnPreNewConnector](#)<sup>[138]</sup>.

## Syntax

**Function** *EA\_OnPostNewConnector(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPostNewConnector* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the new connector: <ul style="list-style-type: none"> <li><i>ConnectorID</i>: A long value corresponding to <i>Connector.ConnectorID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** if the connector has been updated during this notification. Return **False** otherwise.

### 6.5.7.3 EA\_OnPostNewDiagramObject

#### Details

*EA\_OnPostNewDiagramObject* notifies Add-Ins that a new object has been created on a diagram. It enables Add-Ins to modify the object upon creation.

This event occurs after a user has dragged a new object from the Enterprise Architect **Project Browser** or **Resources** window onto a diagram. The notification is provided immediately after the object is added to the diagram. Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA\\_OnPreNewDiagramObject](#)<sup>[139]</sup>.

## Syntax

**Function** *EA\_OnPostNewDiagramObject(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPostNewDiagramObject* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the new element: <ul style="list-style-type: none"> <li><i>ObjectID</i>: A long value corresponding to <i>Object.ObjectID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** if the element has been updated during this notification. Return **False** otherwise.

### 6.5.7.4 EA\_OnPostNewAttribute

#### Details

*EA\_OnPostNewAttribute* notifies Add-Ins that a new attribute has been created on a diagram. It enables Add-Ins to modify the attribute upon creation.

This event occurs when a user creates a new attribute on an element by either drag-dropping from the **Project Browser**, using the **Attributes Properties** dialog, or using the in-place editor on the diagram. The notification is provided immediately after the attribute is created. Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA\\_OnPreNewAttribute](#)<sup>[139]</sup>.

#### Syntax

**Function** *EA\_OnPostNewAttribute*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPostNewAttribute* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the new attribute: <ul style="list-style-type: none"> <li><i>AttributeID</i>: A long value corresponding to <i>Attribute.AttributeID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Return **True** if the attribute has been updated during this notification. Return **False** otherwise.

### 6.5.7.5 EA\_OnPostNewMethod

#### Details

*EA\_OnPostNewMethod* notifies Add-Ins that a new method has been created on a diagram. It enables Add-Ins to modify the method upon creation.

This event occurs when a user creates a new method on an element by either drag-dropping from the **Project Browser**, using the method's **Properties** dialog, or using the in-place editor on the diagram. The notification is provided immediately after the method is created. Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA\\_OnPreNewMethod](#)<sup>[140]</sup>.

#### Syntax

**Function** *EA\_OnPostNewMethod*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPostNewMethod* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the new method: <ul style="list-style-type: none"> <li><i>MethodID</i>: A long value corresponding to <i>Method.MethodID</i>.</li> </ul>



Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** if the method has been updated during this notification. Return **False** otherwise.

### 6.5.7.6 EA\_OnPostNewPackage

#### Details

*EA\_OnPostNewPackage* notifies Add-Ins that a new package has been created on a diagram. It enables Add-Ins to modify the package upon creation.

This event occurs when a user drags a new package from the Enterprise Architect UML **Toolbox** or **Resources** window onto a diagram, or by selecting the **New Package** icon from the **Project Browser**. Set *Repository.SuppressEADialogs* to **true** to suppress Enterprise Architect from showing its default dialogs.

Also look at [EA\\_OnPreNewPackage](#)<sup>[14]</sup>.

#### Syntax

**Function** *EA\_OnPostNewPackage*(*Repository As EA.Repository, Info As EA.EventProperties*) **As Boolean**

The *EA\_OnPostNewPackage* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[20]</sup>	IN	Contains the following <i>EventProperty</i> objects for the new package: <ul style="list-style-type: none"> <li><i>PackageID</i>: A long value corresponding to <i>Package.PackageID</i>.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** if the package has been updated during this notification. Return **False** otherwise.

### 6.5.8 EA\_OnPostInitialized

#### Details

*EA\_OnPostInitialized* notifies Add-Ins that the Repository object has finished loading and any necessary initialization steps can now be performed on the object.

For example, the Add-In can create an **Output** tab using [Repository.CreateOutputTab](#)<sup>[195]</sup>.

#### Syntax

**Sub** *EA\_OnPostInitialized*(*Repository As EA.Repository*)

The *EA\_OnPostInitialized* function syntax contains the following elements.

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

None.

## 6.5.9 EA\_OnPostTransform

### Details

*EA\_OnPostTransform* notifies Add-Ins that an MDG transformation has taken place with the output in the specified target package.

This event occurs when a user runs an MDG transform on one or more target packages. The notification is provided for each transform/target package immediately after all transform processes have completed.

### Syntax

**Function** *EA\_OnPostTransform*(*Repository* As *EA.Repository*, *Info* As *EA.EventProperties*) As *Boolean*

The *EA\_OnPostTransform* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[20]</sup>	IN	Contains the following <i>EventProperty Objects</i> for the transform performed: <ul style="list-style-type: none"> <li><i>Transform</i>: A string value corresponding to the name of the transform used</li> <li><i>PackageID</i>: A long value corresponding to <i>Package.PackageID</i> of the destination package.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An EA.Repository object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Reserved for future use.

## 6.5.10 Technology Events

Enterprise Architect Add-Ins can respond to the following events associated with the use of MDG Technologies:

- [EA\\_OnInitializeTechnologies](#)<sup>[147]</sup>
- [EA\\_OnPreActivateTechnology](#)<sup>[147]</sup>
- [EA\\_OnPostActivateTechnology](#)<sup>[148]</sup>
- [EA\\_OnPreDeleteTechnology](#)<sup>[148]</sup> (Deprecated)
- [EA\\_OnDeleteTechnology](#)<sup>[149]</sup> (Deprecated)
- [EA\\_OnImportTechnology](#)<sup>[149]</sup> (Deprecated)

### 6.5.10.1 EA\_OnInitializeTechnologies

#### Details

*EA\_OnInitializeTechnologies* requests that an Add-In pass an MDG Technology to Enterprise Architect for loading.

This event occurs on Enterprise Architect startup. Return your technology XML to this function and Enterprise Architect loads and enables it.

#### Syntax

**Function** EA\_OnInitializeTechnologies(*Repository As EA.Repository*) As Object

The *EA\_OnInitializeTechnologies* function syntax contains the following element:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Return the MDG Technology as a single XML string.

#### Example

```
Public Function EA_OnInitializeTechnologies(ByVal Repository As EA.Repository) As Object
    EA_OnInitializeTechnologies = My.Resources.MyTechnology
End Function
```

### 6.5.10.2 EA\_OnPreActivateTechnology

#### Details

*EA\_OnPreActivateTechnology* notifies Add-Ins that an MDG Technology resource is about to be activated in the model. This event occurs when a user selects to activate an MDG Technology resource in the model (by clicking on the **Set Active** button on the **MDG Technologies** dialog (see *Extending UML With Enterprise Architect*), or by selecting the technology in the listbox in the **Default Tools** toolbar).

The notification is provided immediately after the user attempts to activate the MDG Technology, so that the Add-In can permit or disable activation of the Technology.

Also look at [EA\\_OnPostActivateTechnology](#)<sup>[148]</sup>.

#### Syntax

**Function** EA\_OnPreActivateTechnology(*Repository As EA.Repository, Info As EA.EventProperties*) As Boolean

The *EA\_OnPreActivateTechnology* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the MDG Technology to be activated: <ul style="list-style-type: none"> <li><i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** to enable activation of the MDG Technology resource in the model. Return **False** to disable activation of the MDG Technology resource.

### 6.5.10.3 EA\_OnPostActivateTechnology

#### Details

*EA\_OnPostActivateTechnology* notifies Add-Ins that an MDG Technology resource has been activated in the model. This event occurs when a user activates an MDG Technology resource in the model (by clicking on the **Set Active** button on the **MDG Technologies** dialog (see *Extending UML With Enterprise Architect*), or by selecting the technology in the listbox in the **Default Tools** toolbar). The notification is provided immediately after the user succeeds in activating the MDG Technology, so that the Add-In can update the Technology if necessary.

Also look at [EA\\_OnPreActivateTechnology](#)<sup>[147]</sup>.

#### Syntax

**Function** *EA\_OnPostActivateTechnology(Repository As EA.Repository, Info As EA.EventProperties)*

The *EA\_OnPostActivateTechnology* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects for the MDG Technology to be activated: <ul style="list-style-type: none"> <li><i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** if the MDG Technology resource is updated during this notification. Return **False** otherwise.

### 6.5.10.4 EA\_OnPreDeleteTechnology

**Deprecated** - refers to deleting a technology through the **Resources** window; this process is no longer recommended. See:

- [EA\\_OnPreActivateTechnology](#)<sup>[147]</sup>
- [EA\\_OnPostActivateTechnology](#)<sup>[148]</sup>
- [EA\\_OnInitializeTechnologies](#)<sup>[147]</sup>.

#### Details

*EA\_OnPreDeleteTechnology* notifies Add-Ins that an MDG Technology resource is about to be deleted from the model. This event occurs when a user deletes an MDG Technology resource from the model. The notification is provided immediately after the user confirms their request to delete the MDG Technology, so that the Add-In can disable deletion of the MDG Technology.

Also look at [EA\\_OnDeleteTechnology](#)<sup>[149]</sup>.

#### Syntax

**Function** *EA\_OnPreDeleteTechnology(Repository As EA.Repository, Info As EA.EventProperties) As Boolean*

The *EA\_OnPreDeleteTechnology* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <small>[207]</small>	IN	Contains the following <i>EventProperty</i> objects for the MDG Technology to be deleted: <ul style="list-style-type: none"> <li><i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.</li> </ul>
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Return **True** to enable deletion of the MDG Technology resource from the model. Return **False** to disable deletion of the MDG Technology resource.

### 6.5.10.5 EA\_OnDeleteTechnology

**Deprecated** - refers to deleting a technology through the **Resources** window; this process is no longer recommended. See:

- [EA\\_OnPreActivateTechnology](#)  
[147]
- [EA\\_OnPostActivateTechnology](#)  
[148]
- [EA\\_OnInitializeTechnologies](#)  
[147].

## Details

*EA\_OnDeleteTechnology* notifies Add-Ins that an MDG Technology resource has been deleted from the model.

This event occurs after a user has deleted an MDG Technology resource from the model. Add-Ins that require an MDG Technology resource to be loaded can catch this event to disable certain functionality.

Also look at [EA\\_OnPreDeleteTechnology](#)  
[145].

## Syntax

**Sub EA\_OnDeleteTechnology(Repository As EA.Repository, Info As EA.EventProperties)**

The *EA\_OnDeleteTechnology* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <small>[207]</small>	IN	Contains the following <i>EventProperty</i> objects: <ul style="list-style-type: none"> <li><i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.</li> </ul>
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

None.

### 6.5.10.6 EA\_OnImportTechnology

**Deprecated** - refers to deleting a technology through the **Resources** window; this process is no longer recommended. See:

- [EA\\_OnPreActivateTechnology](#)  
[147]
- [EA\\_OnPostActivateTechnology](#)  
[148]

- [EA\\_OnInitializeTechnologies](#)<sup>[147]</sup>.

## Details

*EA\_OnImportTechnology* notifies Add-Ins that you have imported an MDG Technology resource into the model.

This event occurs after you have imported an MDG Technology resource into the model. Add-Ins that require an MDG Technology resource to be loaded can catch this Add-In to enable certain functionality.

## Syntax

**Sub *EA\_OnImportTechnology*(*Repository As EA.Repository, Info As EA.EventProperties*)**

The *EA\_OnImportTechnology* function syntax contains the following elements:

Parameter	Type	Direction	Description
Info	<a href="#">EA.EventProperties</a> <sup>[207]</sup>	IN	Contains the following <i>EventProperty</i> objects: <ul style="list-style-type: none"> <li>• <i>TechnologyID</i>: A string value corresponding to the MDG Technology ID.</li> </ul>
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

None.

### 6.5.11 Context Item Events

Enterprise Architect Add-Ins can respond to the following events associated with changing context:

- [EA\\_OnContextItemChanged](#)<sup>[150]</sup>
- [EA\\_OnContextItemDoubleClicked](#)<sup>[151]</sup>
- [EA\\_OnNotifyContextItemModified](#)<sup>[152]</sup>

#### 6.5.11.1 EA\_OnContextItemChanged

## Details

*EA\_OnContextItemChanged* notifies Add-Ins that a different item is now in context.

This event occurs after a user has selected an item anywhere in the Enterprise Architect GUI. Add-Ins that require knowledge of the current item in context can subscribe to this broadcast function. If *ot = otRepository*, then this function behaves the same as [EA\\_FileOpen](#)<sup>[133]</sup>.

Also look at [EA\\_OnContextItemDoubleClicked](#)<sup>[151]</sup> and [EA\\_OnNotifyContextItemModified](#)<sup>[152]</sup>.

## Syntax

**Sub *EA\_OnContextItemChanged*(*Repository As EA.Repository, GUID As String, ot as EA.ObjectType*)**

The *EA\_OnContextItemChanged* function syntax contains the following elements:

Parameter	Type	Direction	Description
GUID	<i>String</i>	IN	Contains the GUID of the new context item. This value corresponds to the following properties, depending on the value of the <i>ot</i> parameter:

Parameter	Type	Direction	Description
			<i>ot (ObjectType)</i> - GUID value <i>otElement</i> - <i>Element.ElementGUID</i> <i>otPackage</i> - <i>Package.PackageGUID</i> <i>otDiagram</i> - <i>Diagram.DiagramGUID</i> <i>otAttribute</i> - <i>Attribute.AttributeGUID</i> <i>otMethod</i> - <i>Method.MethodGUID</i> <i>otConnector</i> - <i>Connector.ConnectorGUID</i> <i>otRepository</i> - NOT APPLICABLE, GUID is an empty string
<b>ot</b>	<i>EA.ObjectType</i>	IN	Specifies the type of the new context item.
<b>Repository</b>	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

None.

### 6.5.11.2 EA\_OnContextItemDoubleClicked

#### Details

*EA\_OnContextItemDoubleClicked* notifies Add-Ins that the user has double-clicked the item currently in context.

This event occurs when a user has double-clicked (or pressed **[Enter]**) on the item in context, either in a diagram or in the **Project Browser**. Add-Ins to handle events can subscribe to this broadcast function.

Also look at [EA\\_OnContextItemChanged](#)[150] and [EA\\_OnNotifyContextItemModified](#)[152].

#### Syntax

**Function** *EA\_OnContextItemDoubleClicked(Repository As EA.Repository, GUID As String, ot as EA.ObjectType)*

The *EA\_OnContextItemDoubleClicked* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>GUID</b>	<i>String</i>	IN	Contains the GUID of the new context item. This value corresponds to the following properties, depending on the value of the <b>ot</b> parameter:  <i>ot (ObjectType)</i> - GUID value <i>otElement</i> - <i>Element.ElementGUID</i> <i>otPackage</i> - <i>Package.PackageGUID</i> <i>otDiagram</i> - <i>Diagram.DiagramGUID</i> <i>otAttribute</i> - <i>Attribute.AttributeGUID</i> <i>otMethod</i> - <i>Method.MethodGUID</i> <i>otConnector</i> - <i>Connector.ConnectorGUID</i>

Parameter	Type	Direction	Description
ot	<i>EA.ObjectType</i>	IN	Specifies the type of the new context item.
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Return **True** to notify Enterprise Architect that the double-click event has been handled by an Add-In. Return **False** to enable Enterprise Architect to continue processing the event.

#### 6.5.11.3 EA\_OnNotifyContextItemModified

##### Details

*EA\_OnNotifyContextItemModified* notifies Add-Ins that the current context item has been modified.

This event occurs when a user has modified the context item. Add-Ins that require knowledge of when an item has been modified can subscribe to this broadcast function.

Also look at [EA\\_OnContextItemChanged](#)[150] and [EA\\_OnContextItemDoubleClicked](#)[151].

##### Syntax

**Sub EA\_OnNotifyContextItemModified(Repository As EA.Repository, GUID As String, ot as EA.ObjectType)**

The *EA\_OnNotifyContextItemModified* function syntax contains the following elements:

Parameter	Type	Direction	Description
GUID	<i>String</i>	IN	Contains the GUID of the new context item. This value corresponds to the following properties, depending on the value of the <b>ot</b> parameter: <ul style="list-style-type: none"> <li><i>ot (ObjectType)</i> - <i>GUID value</i></li> <li><i>otElement</i> - <i>Element.ElementGUID</i></li> <li><i>otPackage</i> - <i>Package.PackageGUID</i></li> <li><i>otDiagram</i> - <i>Diagram.DiagramGUID</i></li> <li><i>otAttribute</i> - <i>Attribute.AttributeGUID</i></li> <li><i>otMethod</i> - <i>Method.MethodGUID</i></li> <li><i>otConnector</i> - <i>Connector.ConnectorGUID</i></li> </ul>
ot	<i>EA.ObjectType</i>	IN	Specifies the type of the new context item.
Repository	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

None.



## 6.5.12 Compartment Events

Enterprise Architect Add-Ins can respond to the following events associated with user-generated element compartments:

- [EA\\_QueryAvailableCompartments](#)<sup>[153]</sup>
- [EA\\_GetCompartmentData](#)<sup>[153]</sup>

### 6.5.12.1 EA\_QueryAvailableCompartments

#### Details

This event occurs when Enterprise Architect's diagrams are refreshed. It is a request for the Add-In to provide a list of user-defined compartments. The [EA\\_GetCompartmentData](#)<sup>[153]</sup> event then queries each object for the data to display in each user-defined compartment.

#### Syntax

**Function** *EA\_QueryAvailableCompartments(Repository As EA.Repository) As Variant*

The *EA\_QueryAvailableCompartments* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

A *String* containing a comma-separated list of user-defined compartments.

#### Example

```
Function EA_QueryAvailableCompartments(Repository As EA.Repository) As Variant
    Dim sReturn As String
    sReturn = ""
    If m_FirstCompartmentVisible = True Then
        sReturn = sReturn + "first,"
    End If
    If m_SecondCompartmentVisible = True Then
        sReturn = sReturn + "second,"
    End If
    If m_ThirdCompartmentVisible = True Then
        sReturn = sReturn + "third,"
    End If

    If Len(sReturn) > 0 Then
        sReturn = Left(sReturn, Len(sReturn)-1)
    End If

    EA_QueryAvailableCompartments = sReturn
End Function
```

### 6.5.12.2 EA\_GetCompartmentData

#### Details

This event occurs when Enterprise Architect is instructed to redraw an element. It requests that the Add-In provide the data to populate the element's compartment.

#### Syntax

**Function** *EA\_GetCompartmentData(Repository As EA.Repository, sCompartment As String, sGUID As*

**String, oType As EA.ObjectType) As Variant**

The *EA\_QueryAvailableCompartments* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>oType</b>	<i>ObjectType</i>	IN	The type of the element for which data is being requested.
<b>Repository</b>	<a href="#">EA.Repository</a> [190]	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
<b>sCompartment</b>	<i>String</i>	IN	The name of the compartment for which data is being requested.
<b>sGUID</b>	<i>String</i>	IN	The GUID of the element for which data is being requested.

**Return Value**

Variant containing a formatted string. See the example below to understand the format.

**Example**

Function *EA\_GetCompartmentData*(Repository As EA.Repository, sCompartment As String, sGUID As String, oType As EA.ObjectType) As Variant

```
If Repository Is Nothing Then
Exit Function
End If
```

```
Dim sCompartmentData As String
Dim oXML As MSXML2.DOMDocument
Dim Nodes As MSXML2.IXMLDOMNodeList
Dim Node1 As MSXML2.IXMLDOMNode
Dim Node As MSXML2.IXMLDOMNode
Dim sData As String
```

```
sCompartmentData = ""
Set oXML = New MSXML2.DOMDocument
sData = ""
```

```
On Error GoTo ERR_GetCompartmentData
```

```
oXML.loadXML (Repository.GetTreeXMLByGUID(sGUID))
Set Node1 = oXML.selectSingleNode("//ModelItem")
```

```
If Node1 Is Nothing Then
Exit Function
End If
```

```
sCompartmentData = sCompartmentData + "Name=" + sCompartment + ";"
sCompartmentData = sCompartmentData + "OwnerGUID=" + sGUID + ";"
sCompartmentData = sCompartmentData + "Options=SkipIfOnDiagram&_eq_^1&_sc_^"
```

```
Select Case sCompartment
```

```
Case "parts"
```

```
Set Nodes = Node1.selectNodes("ModelItem[@Metatype=""Part""]")
```

```
For Each Node In Nodes
```

```
sData = sData + "Data&_eq_^" + Node.Attributes.getNamedItem("Name").nodeValue + "&_sc_^"
```

```
sData = sData + "GUID&_eq_^" + Node.Attributes.getNamedItem("GUID").nodeValue + "&_sc_^"
```

```
Next
```

```
Case "ports"
```

```
Set Nodes = Node1.selectNodes("ModelItem[@Metatype=""Port""]")
```

```
For Each Node In Nodes
```

```
sData = sData + "Data&_eq_^" + Node.Attributes.getNamedItem("Name").nodeValue + "&_sc_^"
```

```
sData = sData + "GUID&_eq_^" + Node.Attributes.getNamedItem("GUID").nodeValue + "&_sc_^"
```

```
Next
```

```
End Select
```

```

' If there's no data to display, then don't return any compartment data
If sData <> "" Then
    sCompartmentData = sCompartmentData + "CompartmentData=" + sData + ";"
Else
    sCompartmentData = ""
End If

EA_GetCompartmentData = sCompartmentData
Exit Function

ERR_GetCompartmentData:
EA_GetCompartmentData = ""

End Function

```

### 6.5.13 Model Validation Broadcasts

#### Perform Model Validation from an Add-In

Using Enterprise Architect broadcasts, it is possible to define a set of rules that are evaluated when the user instructs Enterprise Architect to perform model validation. An Add-In that performs model validation would involve the following broadcast events:

- [EA\\_OnInitializeUserRules](#)<sup>[155]</sup> is intercepted in order to define rule categories and rules.
- [EA\\_OnStartValidation](#)<sup>[156]</sup> can be intercepted to perform any required processing prior to validation.
- The following functions intercept each request to validate an individual element, package, diagram, connector, attribute and method:
  - [EA\\_OnRunElementRule](#)<sup>[156]</sup>
  - [EA\\_OnRunPackageRule](#)<sup>[157]</sup>
  - [EA\\_OnRunDiagramRule](#)<sup>[157]</sup>
  - [EA\\_OnRunConnectorRule](#)<sup>[158]</sup>
  - [EA\\_OnRunAttributeRule](#)<sup>[158]</sup>
  - [EA\\_OnRunMethodRule](#)<sup>[159]</sup>
- [EA\\_OnEndValidation](#)<sup>[156]</sup> can be intercepted to perform any required clean-up after validation has completed.

Also consider the [Model Validation Example](#)<sup>[160]</sup>.

#### 6.5.13.1 EA\_OnInitializeUserRules

##### Details

*EA\_OnInitializeUserRules* is called on Enterprise Architect start-up and requests that the Add-In provide Enterprise Architect with a rule category and list of rule IDs for model validation.

This function must be implemented by any Add-In that is to perform its own model validation. It must call *Project.DefineRuleCategory* once and *Project.DefineRule* for each rule; these functions are described in the [Project Interface](#)<sup>[262]</sup> section.

##### Syntax

###### Sub EA\_OnInitializeUserRules(*Repository As EA.Repository*)

The *EA\_OnInitializeUserRules* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### 6.5.13.2 EA\_OnStartValidation

#### Details

*EA\_OnStartValidation* notifies Add-Ins that a user has invoked the model validation command from Enterprise Architect.

#### Syntax

**Sub EA\_OnStartValidation(Repository As EA.Repository, ParamArray Args() as Variant)**

The *EA\_OnStartValidation* function syntax contains the following elements:

Parameter	Type	Direction	Description
Args	<i>ParamArray of Variant</i>	IN	Contains a list of Rule Categories that are active for the current invocation of model validation.
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### 6.5.13.3 EA\_OnEndValidation

#### Details

*EA\_OnEndValidation* notifies Add-Ins that model validation has completed. Use this event to arrange any clean-up operations arising from the validation.

#### Syntax

**Sub EA\_OnEndValidation(Repository As EA.Repository, ParamArray Args() as Variant)**

The *EA\_OnEndValidation* function syntax contains the following elements:

Parameter	Type	Direction	Description
Args	<i>ParamArray of Variant</i>	IN	Contains a list of Rule Categories that were active for the invocation of model validation that has just completed.
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### 6.5.13.4 EA\_OnRunElementRule

#### Details

This event is triggered once for each rule defined in *EA\_OnInitializeUserRules* to be performed on each element in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given element, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

Also look at [EA\\_OnInitializeUserRules](#)<sup>[155]</sup>.

#### Syntax

**Sub EA\_OnRunElementRule(Repository As EA.Repository, RuleID As String, Element As EA.Element)**

The *EA\_OnRunElementRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
Element	<i>EA.Element</i>	IN	The element to potentially perform validation on.
Repository	<a href="#">EA.Repository</a>  190	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	<i>String</i>	IN	The ID that was passed into the <i>Project.DefineRule</i> command.

### 6.5.13.5 EA\_OnRunPackageRule

#### Details

This event is triggered once for each rule defined in [EA\\_OnInitializeUserRules](#)<sup>|155|</sup> to be performed on each package in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given package, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

#### Syntax

**Sub EA\_OnRunPackageRule(Repository As EA.Repository, RuleID As String, PackageID As Long)**

The *EA\_OnRunElementRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
PackageID	<i>Long</i>	IN	The ID of the package to potentially perform validation on. Use the <i>Repository.GetPackageByID</i> method to retrieve the package object.
Repository	<a href="#">EA.Repository</a>  190	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	<i>String</i>	IN	The ID that was passed into the <i>Project.DefineRule</i> method.

### 6.5.13.6 EA\_OnRunDiagramRule

#### Details

This event is triggered once for each rule defined in [EA\\_OnInitializeUserRules](#)<sup>|155|</sup> to be performed on each diagram in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given diagram, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

#### Syntax

**Sub EA\_OnRunDiagramRule(Repository As EA.Repository, RuleID As String, DiagramID As Long)**

The *EA\_OnRunDiagramRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
DiagramID	<i>Long</i>	IN	The ID of the diagram to potentially perform validation on. Use the <i>Repository.GetDiagramByID</i> method to retrieve the diagram object.
Repository	<a href="#">EA.Repository</a>  190	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

Parameter	Type	Direction	Description
RuleID	String	IN	The ID that was passed into the <i>Project.DefineRule</i> command.

### 6.5.13.7 EA\_OnRunConnectorRule

#### Details

This event is triggered once for each rule defined in [EA\\_OnInitializeUserRules](#)<sup>[155]</sup> to be performed on each connector in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given connector, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

#### Syntax

**Sub EA\_OnRunConnectorRule(Repository As EA.Repository, RuleID As String, ConnectorID As Long)**

The *EA\_OnRunConnectorRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
ConnectorID	Long	IN	The ID of the connector to potentially perform validation on. Use the <i>Repository.GetConnectorByID</i> method to retrieve the connector object.
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String	IN	The ID that was passed into the <i>Project.DefineRule</i> command.

### 6.5.13.8 EA\_OnRunAttributeRule

#### Details

This event is triggered once for each rule defined in [EA\\_OnInitializeUserRules](#)<sup>[155]</sup> to be performed on each attribute in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given attribute, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

#### Syntax

**Sub EA\_OnRunAttributeRule(Repository As EA.Repository, RuleID As String, AttributeGUID As String, ObjectID As Long)**

The *EA\_OnRunAttributeRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
AttributeGUID	String	IN	The GUID of the attribute to potentially perform validation on. Use the <i>Repository.GetAttributeByGuid</i> method to retrieve the attribute object.
ObjectID	Long	IN	The ID of the object that owns the given attribute. Use the <i>Repository.GetObjectByID</i> method to retrieve the object.
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String	IN	The ID that was passed into the <i>Project.DefineRule</i>

Parameter	Type	Direction	Description
			command.

### 6.5.13.9 EA\_OnRunMethodRule

#### Details

This event is triggered once for each rule defined in [EA\\_OnInitializeUserRules](#)<sup>[155]</sup> to be performed on each method in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given method, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

#### Syntax

**Sub EA\_OnRunMethodRule(Repository As EA.Repository, RuleID As String, MethodGUID As String, ObjectID As Long)**

The *EA\_OnRunMethodRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>MethodGUID</b>	<i>String</i>	IN	The GUID of the method to potentially perform validation on. Use the <i>Repository.GetMethodByGuid</i> method to retrieve the method object.
<b>ObjectID</b>	<i>Long</i>	IN	The ID of the object that owns the given method. Use the <i>Repository.GetObjectByID</i> method to retrieve the object.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
<b>RuleID</b>	<i>String</i>	IN	The ID that was passed into the <i>Project.DefineRule</i> command.

### 6.5.13.10 EA\_OnRunParameterRule

#### Details

This event is triggered once for each rule defined in [EA\\_OnInitializeUserRules](#)<sup>[155]</sup> to be performed on each parameter in the selection being validated. If you don't want to perform the rule defined by **RuleID** on the given parameter, then simply return without performing any action. On performing any validation, if a validation error is found, use the *Repository.ProjectInterface.PublishResult* method to notify Enterprise Architect.

#### Syntax

**Sub EA\_OnRunParameterRule(Repository As EA.Repository, RuleID As String, ParameterGUID As String, MethodGUID As String, ObjectID As Long)**

The *EA\_OnRunMethodRule* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>MethodGUID</b>	<i>String</i>	IN	The GUID of the method that owns the given parameter. Use the <i>Repository.GetMethodByGuid</i> method to retrieve the method object.
<b>ObjectID</b>	<i>Long</i>	IN	The ID of the object that owns the given parameter. Use the <i>Repository.GetObjectByID</i> method to retrieve the object.
<b>ParameterGUID</b>	<i>String</i>	IN	The GUID of the parameter to potentially perform validation

Parameter	Type	Direction	Description
D			on. Use it to retrieve the parameter by iterating through the <i>Method.Parameters</i> collection.
Repository	<a href="#">EA.Repository</a> [190]	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
RuleID	String	IN	The ID that was passed into the <i>Project.DefineRule</i> command.

### 6.5.13.11 Model Validation Example

The following example code is written in C# and provides a skeleton model validation implementation that you might like to use as a starting point in writing your own model validation rules.

#### Main.cs

```
using System;

namespace myAddin
{
    public class Main
    {
        public Rules theRules;

        public Main()
        {
            theRules = new Rules();
        }

        public string EA_Connect(EA.Repository Repository)
        {
            return "";
        }

        public void EA_Disconnect()
        {
            GC.Collect();
            GC.WaitForPendingFinalizers();
        }

        private bool IsProjectOpen(EA.Repository Repository)
        {
            try
            {
                EA.Collection c = Repository.Models;
                return true;
            }
            catch
            {
                return false;
            }
        }

        public object EA_GetMenuItems(EA.Repository Repository, string MenuLocation, string MenuName)
        {
            switch (MenuName)
            {
                case "":
                    return "-&myAddin";
                case "-&myAddin":
                    string[] ar = { "&Test" };
                    return ar;
            }
            return "";
        }

        public void EA_GetMenuState(EA.Repository Repository, string MenuLocation, string MenuName, string
```



```

ItemName, ref bool IsEnabled, ref bool IsChecked)
{
    // if no open project, disable all menu options
    if (IsProjectOpen(Repository))
        IsEnabled = true;
    else
        IsEnabled = false;
}

public void EA_MenuClick(EA.Repository Repository, string MenuLocation, string MenuName, string
ItemName)
{
    switch (ItemName)
    {
        case "&Test":
            DoTest(Repository);
            break;
    }
}

public void EA_OnInitializeUserRules(EA.Repository Repository)
{
    if (Repository != null)
    {
        theRules.ConfigureCategories(Repository);
        theRules.ConfigureRules(Repository);
    }
}

public void EA_OnRunElementRule(EA.Repository Repository, string RuleID, EA.Element element)
{
    theRules.RunElementRule(Repository, RuleID, element);
}

public void EA_OnRunDiagramRule(EA.Repository Repository, string RuleID, long IDiagramID)
{
    theRules.RunDiagramRule(Repository, RuleID, IDiagramID);
}

public void EA_OnRunConnectorRule(EA.Repository Repository, string RuleID, long IConnectorID)
{
    theRules.RunConnectorRule(Repository, RuleID, IConnectorID);
}

public void EA_OnRunAttributeRule(EA.Repository Repository, string RuleID, string AttGUID, long
IObjectID)
{
    return;
}

public void EA_OnDeleteTechnology(EA.Repository Repository, EA.EventProperties Info)
{
    return;
}

public void EA_OnImportTechnology(EA.Repository Repository, EA.EventProperties Info)
{
    return;
}

private void DoTest(EA.Repository Rep)
{
    // TODO: insert test code here
}
}
}

```

## Rules.cs

```

using System;
using System.Collections;

namespace myAddin
{

```

```

public class Rules
{
    private string m_sCategoryID;
    private System.Collections.ArrayList m_RuleIDs;
    private System.Collections.ArrayList m_RuleIDEx;

    private const string cRule01 = "Rule01";
    private const string cRule02 = "Rule02";
    private const string cRule03 = "Rule03";
    // TODO: expand this list as much as necessary

    public Rules()
    {
        m_RuleIDs = new System.Collections.ArrayList();
        m_RuleIDEx = new System.Collections.ArrayList();
    }

    private string LookupMap(string sKey)
    {
        return DoLookupMap(sKey, m_RuleIDs, m_RuleIDEx);
    }

    private string LookupMapEx(string sRule)
    {
        return DoLookupMap(sRule, m_RuleIDEx, m_RuleIDs);
    }

    private string DoLookupMap(string sKey, ArrayList arrValues, ArrayList arrKeys)
    {
        if (arrKeys.Contains(sKey))
            return arrValues[arrKeys.IndexOf(sKey)].ToString();
        else
            return "";
    }

    private void AddToMap(string sRuleID, string sKey)
    {
        m_RuleIDs.Add(sRuleID);
        m_RuleIDEx.Add(sKey);
    }

    private string GetRuleStr(string sRuleID)
    {
        switch (sRuleID)
        {
            case cRule01:
                return "Error Message 01";
            case cRule02:
                return "Error Message 02";
            case cRule03:
                return "Error Message 03";
            // TODO: add extra cases as much as necessary
        }
        return "";
    }

    public void ConfigureCategories(EA.Repository Repository)
    {
        EA.Project Project = Repository.GetProjectInterface();
        m_sCategoryID = Project.DefineRuleCategory("Enterprise Collaboration Architecture (ECA
Rules)");
    }

    public void ConfigureRules(EA.Repository Repository)
    {
        EA.Project Project = Repository.GetProjectInterface();
        AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrType.mvError,
GetRuleStr(cRule01)), cRule01);
        AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrType.mvError,
GetRuleStr(cRule02)), cRule02);
        AddToMap(Project.DefineRule(m_sCategoryID, EA.EnumMVErrType.mvError,
GetRuleStr(cRule03)), cRule03);
        // TODO: expand this list
    }
}

```

```

public void RunConnectorRule(EA.Repository Repository, string sRuleID, long IConnectorID)
{
    EA.Connector Connector = Repository.GetConnectorByID((int)IConnectorID);
    if (Connector != null)
    {
        switch (LookupMapEx(sRuleID))
        {
            case cRule02:
                // TODO: perform rule 2 check
                break;
            // TODO: add more cases
        }
    }
}

public void RunDiagramRule(EA.Repository Repository, string sRuleID, long IDiagramID)
{
    EA.Diagram Diagram = Repository.GetDiagramByID((int)IDiagramID);
    if (Diagram != null)
    {
        switch (LookupMapEx(sRuleID))
        {
            case cRule03:
                // TODO: perform rule 3 check
                break;
            // TODO: add more cases
        }
    }
}

public void RunElementRule(EA.Repository Repository, string sRuleID, EA.Element Element)
{
    if (Element != null)
    {
        switch (LookupMapEx(sRuleID))
        {
            case cRule01:
                DoRule01(Repository, Element);
                break;
            // TODO: add more cases
        }
    }
}

private void DoRule01(EA.Repository Repository, EA.Element Element)
{
    if (Element.Stereotype != "myStereotype")
        return;

    // TODO: validation logic here

    // report validation errors
    EA.Project Project = Repository.GetProjectInterface();
    Project.PublishResult(LookupMap(cRule01), EA.EnumMVErrorType.mvError,
    GetRuleStr(cRule01));
}
}
}

```

### 6.5.14 EA\_OnRetrieveModelTemplate

#### Details

*EA\_OnRetrieveModelTemplate* requests that an Add-In pass a model template to Enterprise Architect.

This event occurs when a user executes the **Add a New Model Using Wizard** command to add a model that has been defined by an MDG Technology. See the [Incorporate Model Templates in a Technology](#)<sup>[57]</sup> topic for details of how to define such model templates.

#### Syntax

**Function** EA\_OnRetrieveModelTemplate(Repository As EA.Repository,sLocation As String) As String

The *EA\_OnRetrieveModelTemplate* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
sLocation	String	IN	The name of the template requested. This should match the <i>location</i> attribute in the <i>&lt;ModelTemplates&gt;</i> section of an MDG Technology File. For more information, see the <i>Incorporate Model Templates in a Technology</i> topic.

## Return Value

Return a string containing the XML export of the model that is being used as a template.

## Example

```
Public Function EA_OnRetrieveModelTemplate(ByRef Rep As EA.Repository, ByRef sLocation As String) As String
    Dim sTemplate As String
    Select Case sLocation
        Case "Templates\Template1.xml"
            sTemplate = My.Resources.Template1
        Case "Templates\Template2.xml"
            sTemplate = My.Resources.Template2
        Case "Templates\Template3.xml"
            sTemplate = My.Resources.Template3
        Case Else
            MsgBox("Path for " & sLocation & " not found")
            sTemplate = ""
    End Select
    EA_OnRetrieveModelTemplate = sTemplate
End Function
```

## 6.6 Custom Views

Enterprise Architect enables custom windows to be inserted as tabs in the **Diagram View** that appears at the center of the Enterprise Architect frame.

[Creating a custom view](#)<sup>[164]</sup> enables you to easily and quickly tab between a custom interface and diagrams and other views normally provided by Enterprise Architect.

Uses for this facility include:

- Reports and graphs showing summary data of the model
- Alternative views of a diagram
- Alternative views of the model
- Views of external data related to model data
- Documentation tools.

### 6.6.1 Create a Custom View

A custom view must be designed as an ActiveX custom control and inserted through the automation interface.

ActiveX custom controls can be created using most well-known programming tools including Microsoft Visual Studio.NET. See the documentation provided by the relevant vendor on how to create a custom control to produce an OCX file.

Once the custom control has been created and registered on the target system, it can be added through the **AddTab()** method of the [Repository](#)<sup>[190]</sup> object.

While it is possible to call **AddTab()** from any automation client, it is likely that you would call it from an Add-In, and that Add-In is defined in the same OCX that provides the custom view.

Example C# code is shown below:

```

public class Addin
{
    UserControl1 m_MyControl;

    public void EA_Connect(EA.Repository Rep)
    {
    }

    public object EA_GetMenuItems(EA.Repository Repository, string Location, string MenuName)
    {
        if( MenuName == "" )
            return "-&C# Control Demo";
        else
        {
            String[] ret = {"&Create", "&Show Button"};
            return ret;
        }
    }

    public void EA_MenuClick(EA.Repository Rep, string Location, string MenuName, string ItemName)
    {
        if( ItemName == "&Create" )
            m_MyControl = (UserControl1) Rep.AddTab("C# Demo","ContDemo.UserControl1");
        else
            m_MyControl.ShowButton();
    }
}

```

## 6.7 MDG Add-Ins

MDG Add-Ins are specialized types of Add-Ins that have additional features and extra requirements for Add-In authors who want to contribute to Enterprise Architect's goal of Model Driven Generation. Unlike general Add-In events, MDG Add-In events are only sent to the Add-In that has taken ownership of an Enterprise Architect model branch on a particular PC.

One of the additional responsibilities of an MDG Add-In is to take ownership of a branch of an Enterprise Architect model, which is done through the [MDG Connect](#)<sup>[166]</sup> event.

MDG Add-Ins identify themselves as such during [EA Connect](#)<sup>[128]</sup> by returning the string *MDG*.

Unlike ordinary Add-Ins, responding to MDG Add-In events is not optional, and methods must be published for each of the [MDG Events](#)<sup>[165]</sup>.

Two examples of MDG Add-Ins are the commercially available *MDG Link for Eclipse* and *MDG Link for Visual Studio*, published by [Sparx Systems](#).

### 6.7.1 MDG Events

An MDG Add-In must respond to all MDG Events. These events usually identify processes such as Build, Run, Synchronize, PreMerge and PostMerge, amongst others.

An MDG Link Add-In is expected to implement some form of forward and reverse engineering capability within Enterprise Architect, and as such requires access to a specific set of events, all to do with generation, synchronization and general processes concerned with converting models to code and code to models.

- [MDG\\_BuildProject](#)<sup>[166]</sup>
- [MDG\\_Connect](#)<sup>[166]</sup>
- [MDG\\_Disconnect](#)<sup>[167]</sup>
- [MDG\\_GetConnectedPackages](#)<sup>[167]</sup>
- [MDG\\_GetProperty](#)<sup>[168]</sup>
- [MDG\\_Merge](#)<sup>[169]</sup>
- [MDG\\_NewClass](#)<sup>[170]</sup>
- [MDG\\_PostGenerate](#)<sup>[171]</sup>
- [MDG\\_PostMerge](#)<sup>[171]</sup>
- [MDG\\_PreGenerate](#)<sup>[172]</sup>

- [MDG\\_PreMerge](#)<sup>[172]</sup>
- [MDG\\_PreReverse](#)<sup>[173]</sup>
- [MDG\\_RunExe](#)<sup>[174]</sup>
- [MDG\\_View](#)<sup>[174]</sup>

### 6.7.1.1 MDGBuild Project

#### Details

*MDG\_BuildProject* enables the Add-In to handle file changes caused by generation. This function is called in response to a user selecting the **Add-Ins | Build Project** menu option.

Respond to this event by compiling the project source files into a running application.

Also look at [MDG\\_RunExe](#)<sup>[174]</sup>.

#### Syntax

**Sub** *MDG\_BuildProject*(*Repository As EA.Repository, PackageGuid As String*)

The *MDG\_BuildProject* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>PackageGuid</b>	<i>String</i>	IN	The GUID identifying the Enterprise Architect package subtree that is controlled by the Add-In.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

None.

### 6.7.1.2 MDGConnect

#### Details

*MDG\_Connect* enables the Add-In to handle user driven request to connect a model branch to an external application. This function is called when the user attempts to connect a particular Enterprise Architect package to an as yet unspecified external project. This event enables the Add-In to interact with the user to specify such a project.

The Add-In is responsible for retaining the connection details, which should be stored on a per-user or per-workstation basis. That is, users who share a common Enterprise Architect model over a network should be able to connect and disconnect to external projects independently of one another.

The Add-In should therefore not store connection details in an Enterprise Architect repository. A suitable place to store such details would be:

```
SHGetFolderPath(..CSIDL_APPDATA..)AddinName.
```

The *PackageGuid* parameter is the same identifier as required for most events relating to the MDG Add-In. Therefore it is recommended that the connection details be indexed using the *PackageGuid* value.

The *PackageID* parameter is provided to aid fast retrieval of package details from Enterprise Architect, should this be required.

Also look at [MDG\\_Disconnect](#)<sup>[167]</sup>.

#### Syntax

**Function** *MDG\_Connect*(*Repository As EA.Repository, PackageID as Long, PackageGuid As String*) **As Long**

The *MDG\_Connect* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>PackageGuid</b>	<i>String</i>	IN	The unique ID identifying the project provided by the Add-In when a connection to a project branch of an Enterprise Architect model was first established.
<b>PackageID</b>	<i>Long</i>	IN	The <i>PackageID</i> of the Enterprise Architect package the user has requested to have connected to an external project.
<b>Repository</b>	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Returns a non-zero to indicate that a connection has been made; a zero indicates that the user has not nominated a project and connection should not proceed.

#### 6.7.1.3 MDGDisconnect

##### Details

*MDG\_Disconnect* enables the Add-In to respond to user requests to disconnect the model branch from an external project. This function is called when the user attempts to disconnect an associated external project. The Add-In is required to delete the details of the connection.

Also look at [MDG\\_Connect](#)<sup>[166]</sup>.

##### Syntax

**Function** *MDG\_Disconnect(Repository As EA.Repository, PackageGuid As String) As Long*

The *MDG\_Disconnect* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>PackageGuid</b>	<i>String</i>	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
<b>Repository</b>	<a href="#">EA.Repository</a> <small>[190]</small>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

### Return Value

Returns a non-zero to indicate that a disconnection has occurred enabling Enterprise Architect to update the user interface. A zero indicates that the user has not disconnected from an external project.

#### 6.7.1.4 MDGGetConnectedPackages

##### Details

*MDG\_GetConnectedPackages* enables the Add-In to return a list of current connection between Enterprise Architect and an external application. This function is called when the Add-In is first loaded, and is expected to return a list of the available connections to external projects for this Add-In.

Also look at [MDG\\_Connect](#)<sup>[166]</sup>.

## Syntax

### Function MDG\_GetConnectedPackages(*Repository As EA.Repository*) As Variant

The *MDG\_GetConnectedPackages* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> [190]	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

Returns an array of GUID strings representing individual Enterprise Architect packages.

### 6.7.1.5 MDGGetProperty

## Details

*MDG\_GetProperty* provides miscellaneous Add-In details to Enterprise Architect. This function is called by Enterprise Architect to poll the Add-In for information relating to the *PropertyName*. This event should occur in as short a duration as possible as Enterprise Architect does not cache the information provided by the function.

Values corresponding to the following *PropertyNames* must be provided:

- **IconID** - Return the name of a DLL and a resource identifier in the format *#ResID*, where the resource ID indicates an Icon; for example, *c:\program files\myapp\myapp.dll#101*
- **Language** - Return the default language that Classes should be assigned when they are created in Enterprise Architect
- **HiddenMenus** - Return one or more values from the *MDGMenus* enumeration to hide menus that do not apply to your Add-In. For example:

```
if( PropertyName == "HiddenMenus" )
    return mgBuildProject + mgRun;
```

## Syntax

### Function MDG\_GetProperty(*Repository As EA.Repository, PackageGuid As String, PropertyName As String*) As Variant

The *MDG\_GetProperty* function syntax contains the following elements:

Parameter	Type	Direction	Description
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
PropertyName	String	IN	The name of the property that is used by Enterprise Architect. See <i>Details</i> for the possible values.
Repository	<a href="#">EA.Repository</a> [190]	IN	An <i>EA.Repository</i> object representing the currently-open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

## Return Value

See *Details*, above.



### 6.7.1.6 MDGMerge

#### Details

*MDG\_Merge* enables the Add-In to jointly handle changes to both the model branch and the code project that the model branch is connected to. This event should be called whenever the user has asked to merge their model branch with its connected code project, or whenever the user has established a new connection to a code project. The purpose of this event is to enable the Add-In to interact with the user to perform a merge between the model branch and the connected project.

Also look at [MDG\\_Connect](#)<sup>[166]</sup>, [MDG\\_PreMerge](#)<sup>[172]</sup> and [MDG\\_PostMerge](#)<sup>[171]</sup>.

#### Syntax

**Function** *MDG\_Merge*(*Repository As EA.Repository, PackageGuid As String, SynchObjects As Variant, SynchType As String, ExportObjects As Variant, ExportFiles As Variant, ImportFiles As Variant, IgnoreLocked As String, Language As String*) *As Long*

The *MDG\_Merge* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>ExportFiles</b>	<i>Variant</i>	OUT	A string array containing the list of files for each model object chosen for export by the Add-In. Each entry in this array must have a corresponding entry in the <i>ExportObjects</i> parameter at the same array index, so <i>ExportFiles(2)</i> must contain the filename of the object by <i>ExportObjects(2)</i> .
<b>ExportObjects</b>	<i>Variant</i>	OUT	The string array containing the list of new model objects (in <i>Object ID</i> format) to be exported by Enterprise Architect to the code project.
<b>IgnoreLocked</b>	<i>String</i>	OUT	A value indicating whether to ignore any files locked by the code project (that is, "TRUE" or "FALSE").
<b>ImportFiles</b>	<i>Variant</i>	OUT	A string array containing the list of code files made available to the code project to be newly imported to the model. Enterprise Architect imports each file listed in this array for import into the connected model branch.
<b>Language</b>	<i>String</i>	OUT	The string value containing the name of the code language supported by the code project connected to the model branch.
<b>PackageGuid</b>	<i>String</i>	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[196]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
<b>SynchObjects</b>	<i>Variant</i>	OUT	A string array containing a list of objects ( <i>Object ID</i> format) to be jointly synchronized between the model branch and the project. See <a href="#">below</a> <sup>[170]</sup> for the format of the Object IDs.
<b>SynchType</b>	<i>String</i>	OUT	The value determining the user-selected type of synchronization to take place. See <a href="#">below</a> <sup>[170]</sup> for a list of valid values.

#### Return Value

Return a non-zero if the merge operation completed successfully and a zero value when the operation has been unsuccessful.

#### Merge

A merge consists of three major operations:

- **Export:** Where newly created model objects are exported into code and made available to the code project.
- **Import:** Where newly created code objects, Classes and such things are imported into the model.
- **Synchronize:** Where objects available both to the model and in code are jointly updated to reflect changes made in either the model, code project or both.

## Synchronize Type

The *Synchronize* operation can take place in one of four different ways. Each of these ways corresponds to a value returned by *SynchType*:

- None: (*SynchType* = 0) No synchronization is to be performed
- Forward: (*SynchType* = 1) Forward synchronization, between the model branch and the code project is to occur
- Reverse: (*SynchType* = 2) Reverse synchronization, between the code project and the model branch is to occur
- Both: (*SynchType* = 3) Reverse, then Forward synchronization's are to occur.

## Object ID Format

Each of the Object IDs listed in the string arrays described above should be composed in the following format:  
 (@namespace)\*(#class)\*(\$attribute|%operation|:property)\*

### 6.7.1.7 MDGNewClass

#### Details

*MDG\_NewClass* enables the Add-In to alter details of a Class before it is created.

This method is called when Enterprise Architect generates a new Class, and requires information relating to assigning the language and file path. The file path should be passed back as a return value and the language should be passed back via the language parameter.

Also look at [MDG\\_PreGenerate](#)<sup>[172]</sup>.

#### Syntax

**Function** *MDG\_NewClass*(*Repository As EA.Repository, PackageGuid As String, CodeID As String, Language As String*) *As String*

The *MDG\_NewClass* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>CodeID</b>	<i>String</i>	IN	A string used to identify the code element before it is created, for more information see <a href="#">MDG_View</a> <sup>[174]</sup> .
<b>Language</b>	<i>String</i>	OUT	A string used to identify the programming language for the new Class. The language must be supported by Enterprise Architect.
<b>PackageGuid</b>	<i>String</i>	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[196]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Returns a string containing the file path that should be assigned to the Class.

### 6.7.1.8 MDGPostGenerate

#### Details

*MDG\_PostGenerate* enables the Add-In to handle file changes caused by generation.

This event is called after Enterprise Architect has prepared text to replace the existing contents of a file. Responding to this event enables the Add-In to write to the linked application's user interface rather than modify the file directly.

When the contents of a file are changed, Enterprise Architect passes *FileContents* as a non-empty string. New files created as a result of code generation are also sent through this mechanism, enabling Add-Ins to add new files to the linked project's file list.

When new files are created Enterprise Architect passes *FileContents* as an empty string. When a non-zero is returned by this function, the Add-In has successfully written the contents of the file. A zero value for the return indicates to Enterprise Architect that the file must be saved.

Also look at [MDG\\_PreGenerate](#)<sup>[172]</sup>.

#### Syntax

**Function** *MDG\_PostGenerate*(*Repository* As *EA.Repository*, *PackageGuid* As *String*, *FilePath* As *String*, *FileContents* As *String*) As *Long*

The *MDG\_PostGenerate* function syntax contains the following elements:

Parameter	Type	Direction	Description
<b>FileContents</b>	<i>String</i>	IN	A string containing the proposed contents of the file.
<b>FilePath</b>	<i>String</i>	IN	The path of the file Enterprise Architect intends to overwrite.
<b>PackageGuid</b>	<i>String</i>	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
<b>Repository</b>	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.

#### Return Value

Return value depends on the type of event that this function is responding to (see **Details**, above). This function is required to handle two separate and distinct cases.

### 6.7.1.9 MDGPostMerge

#### Details

*MDG\_PostMerge* is called after a merge process has been completed.

This function is called by Enterprise Architect after the merge process has been completed.

#### Note:

File save checking should not be performed with this function, but should be handled by [MDG\\_PreGenerate](#)<sup>[172]</sup>, [MDG\\_PostGenerate](#)<sup>[171]</sup> and [MDG\\_PreReverse](#)<sup>[173]</sup>.

Also look at [MDG\\_PreMerge](#)<sup>[172]</sup> and [MDG\\_Merge](#)<sup>[169]</sup>.

#### Syntax

**Function** *MDG\_PostMerge*(*Repository* As *EA.Repository*, *PackageGuid* As *String*) As *Long*

The *MDG\_PostMerge* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> [190]	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.

### Return Value

Return a zero value if the post-merge process has failed, a non-zero return indicates that the post-merge has been successful. Enterprise Architect assumes a non-zero return if this method is not implemented

#### 6.7.1.10 MDGPreGenerate

##### Details

*MDG\_PreGenerate* enables the Add-In to deal with unsaved changes. This function is called immediately before Enterprise Architect attempts to generate files from the model. A possible use of this function would be to prompt the user to save unsaved source files.

Also look at [MDG\\_PostGenerate](#)<sup>[17]</sup>.

##### Syntax

**Function** *MDG\_PreGenerate(Repository As EA.Repository, PackageGuid As String) As Long*

The *MDG\_PreGenerate* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> [190]	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.

### Return Value

Return a zero value to abort generation. Any other value enables the generation to continue.

#### 6.7.1.11 MDGPreMerge

##### Details

*MDG\_PreMerge* is called after a merge process has been initiated by the user and before Enterprise Architect performs the merge process.

This event is called after a user has performed their interactions with the merge screen and has confirmed the merge with the **OK** button, but before Enterprise Architect performs the merge process using the data provided by the *MDG\_Merge* call, before any changes have been made to the model or the connected project.

This event is made available to provide the Add-In with the opportunity to generally set internal Add-In flags to augment the *MDG\_PreGenerate*, *MDG\_PostGenerate* and *MDG\_PreReverse* events.

**Note:**

File save checking should not be performed with this function, but should be handled by [MDG\\_PreGenerate](#)<sup>[172]</sup>, [MDG\\_PostGenerate](#)<sup>[171]</sup> and [MDG\\_PreReverse](#)<sup>[173]</sup>.

Also look at [MDG\\_Merge](#)<sup>[169]</sup> and [MDG\\_PostMerge](#)<sup>[171]</sup>.

**Syntax**

**Function** MDG\_PreMerge(*Repository As EA.Repository, PackageGuid As String*) As Long

The MDG\_PreMerge function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.

**Return Value**

A return value of zero indicates that the merge process will not occur. If the value is not zero the merge process will proceed. If this method is not implemented then it is assumed that a merge process is used.

**6.7.1.12 MDGPreReverse****Details**

MDG\_PreReverse enables the Add-In to save file changes before being imported into Enterprise Architect.

This function operates on a list of files that are about to be reverse-engineered into Enterprise Architect. If the user is working on unsaved versions of these files in an editor, you could either prompt the user or save automatically.

Also look at [MDG\\_PostGenerate](#)<sup>[171]</sup> and [MDG\\_PreGenerate](#)<sup>[172]</sup>.

**Syntax**

**Sub** MDG\_PreReverse(*Repository As EA.Repository, PackageGuid As String, FilePaths As Variant*)

The MDG\_PreReverse function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
FilePaths	String array	IN	An array of filepaths pointed to the files that are to be reverse engineered.

**Return Value**

None.

### 6.7.1.13 MDGRunExe

#### Details

*MDG\_RunExe* enables the Add-In to run the target application. This function is called when the user selects the **Add-Ins | Run Exe** menu option. Respond to this event by launching the compiled application.

Also look at [MDG\\_BuildProject](#)<sup>[166]</sup>.

#### Syntax

**Sub MDG\_RunExe(Repository As EA.Repository, PackageGuid As String)**

The *MDG\_RunExe* function syntax contains the following elements:

Parameter	Type	Direction	Description
Repository	<a href="#">EA.Repository</a> <sup>[190]</sup>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.

#### Return Value

None.

### 6.7.1.14 MDGView

#### Details

*MDG\_View* enables the Add-In to display user specified code elements. This function is called by Enterprise Architect when the user asks to view a particular code element. This enables the Add-In to present that element in its own way, usually in a code editor.

#### Syntax

**Function MDG\_View(Repository As EA.Repository, PackageGuid As String, CodeID as String) As Long**

The *MDG\_View* function syntax contains the following elements:

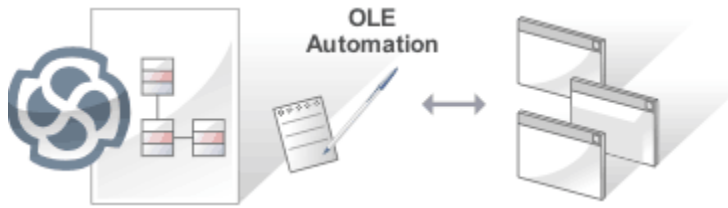
Parameter	Type	Direction	Description
Repository	<i>EA.Repository</i>	IN	An <i>EA.Repository</i> object representing the currently open Enterprise Architect model. Poll its members to retrieve model data and user interface status information.
PackageGuid	String	IN	The GUID identifying the Enterprise Architect package sub-tree that is controlled by the Add-In.
CodeID	String	IN	Identifies the code element in the following format: <type>ElementPart<type>ElementPart... where each element is proceeded with a token identifying its type: @ -namespace # - Class \$ - attribute % - operation For example if a user has selected the <i>m_Name</i> attribute of

Parameter	Type	Direction	Description
			<i>Class1</i> located in <i>namespace Name1</i> , the <i>class ID</i> would be passed through in the following format: @Name1#Class1%m_Name

### Return Value

Return a non-zero value to indicate that the Add-In has processed the request. Returning a zero value results in Enterprise Architect employing the standard viewing process which is to launch the associated source file.

## 7 Enterprise Architect Object Model



### Introduction

Automation provides a way for other applications to access the information in an Enterprise Architect model using Windows OLE Automation (ActiveX). Typically this involves scripting clients such as MS Word or Visual Basic, or using scripts created within Enterprise Architect using the **Scripter** window (see *Using Enterprise Architect - UML Modeling Tool*).

The *Automation Interface* provides a way of accessing the internals of Enterprise Architect models. Examples of things you can do using the Automation Interface include:

- Perform repetitive tasks, such as update the version number for all elements in a model
- Generate code from a State Machine diagram
- Produce custom reports
- Perform ad hoc queries.

### Connecting to the Automation Interface

All development environments capable of generating *ActiveX Com* clients should be able to connect to the Enterprise Architect Automation Interface. This guide provides detailed instructions on [connecting to the interface](#)<sup>[176]</sup> using Microsoft Visual Basic 6.0, Borland Delphi 7.0, Microsoft C# and Java. There are also more detailed steps on how to [set-up Visual Basic](#)<sup>[178]</sup>; the principles are applicable to other languages.

### Examples and Tips

Instruction on how to use the Automation Interface is provided by means of sample code. See [pointers to the samples](#)<sup>[179]</sup> and other [available resources](#)<sup>[181]</sup>. Also, consult the extensive [Reference Section](#)<sup>[181]</sup>.

### Calling Executables from Enterprise Architect

Enterprise Architect can be set up to call an external application. You can pass parameters on the current position selected in the **Project Browser** to the application being called. For instructions, go to the [Call from Enterprise Architect](#)<sup>[180]</sup> topic. A more sophisticated method is to create [Add-Ins](#)<sup>[121]</sup>, which are discussed in a separate topic.

## 7.1 Using the Automation Interface

This section provides instructions on how to connect to and use the Automation Interface, including:

- [Connecting to the Interface](#)<sup>[176]</sup>
- [Set References In Visual Basic](#)<sup>[178]</sup>
- [Examples and Tips](#)<sup>[179]</sup>

### 7.1.1 Connect to the Interface

All development environments capable of generating ActiveX Com clients should be able to connect to the Enterprise Architect Automation Interface.

By way of example, the following sections describe how to connect using several such tools. The procedure might vary slightly with different versions of these products.



## Microsoft Visual Basic 6.0

1. Create a new project.
2. Select the **Project | References** menu option.
3. Select *Enterprise Architect Object Model 2.0* from the list. (If this does not appear, go to the command line and re-register Enterprise Architect using

```
EA.exe /unregister
```

then

```
EA.exe /register).
```

4. See the general library documentation on the use of Classes. The following example creates and opens a repository object:

```
Public Sub ShowRepository()
    Dim MyRep As New EA.Repository
    MyRep.OpenFile "c:\eatest.eap"
End Sub
```

## Borland Delphi 7.0

1. Create a new project.
2. Select the **Project | Import Type Library** menu option.
3. Select **Enterprise Architect Object Model 2.0** from the list. (If this does not appear, go to the command line and re-register Enterprise Architect using

```
EA.exe /unregister
```

then

```
EA.exe /register).
```

4. Click on the **Create Unit** button.
5. Include *EA\_TLB* in Project1's *Uses* clause.
6. See the general library documentation on the use of Classes. The following example creates and opens a repository object:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    r: TRepository;
    b: boolean;
begin
    r:= TRepository.Create(nil);
    b:= r.OpenFile('c:\eatest.eap');
end;
```

## Microsoft C#

1. Select the Visual Studio **Project | Add Reference** menu option.
2. Click on the **Browse** tab.
3. Navigate to the folder in which you installed Enterprise Architect (usually Program Files/Sparx Systems/EA) and select Interop.EA.dll.
4. See the general library documentation on the use of Classes. The following example creates and opens a repository object:

```
private void button1_Click(object sender, System.EventArgs e)
{
    EA.Repository r = new EA.RepositoryClass();
    r.OpenFile("c:\eatest.eap");
}
```

## Java

1. Copy the file SSJavaCOM.dll from the Java API subdirectory of your installed directory (usually Program Files/Sparx Systems/EA) into any location within the Windows PATH. For example, the windows\system32

directory.

- Copy the `eaapi.jar` file from the *Java API* subdirectory of your installed directory (usually Program Files/Sparx Systems/EA) to a location in the Java CLASSPATH or where the Java class loader can find it at run time.
- All of the Classes described in the documentation are in the package `org.sparx`. See the [general library documentation](#)<sup>[18†]</sup> for their use. The following example creates and opens a repository object.

```
public void OpenRepository()
{
    org.sparx.Repository r = new org.sparx.Repository();
    r.OpenFile("c:\\eatest.eap");
}
```

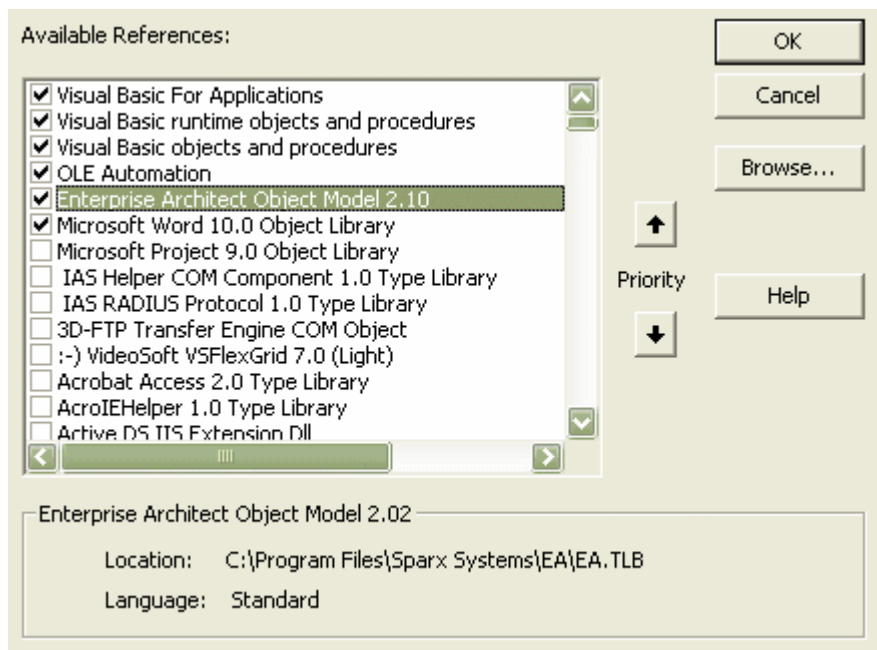
### 7.1.1.1 Set References In Visual Basic

This topic describes how to use the Enterprise Architect ActiveX interface with Visual Basic (VB). Use is ensured for Visual Basic version 6. This might vary slightly with versions other than version 6.

It is assumed that you have accessed VB through a Microsoft Application such as VB 6.0, MS Word or MS Access. If the code is not called from within Word, the *Word VB* reference must also be set.

On creating a new VB project, set a reference to an Enterprise Architect Type Library and a Word Type Library. Follow the steps below:

- Select the **Tools | References** menu option. The following dialog displays:



- Select the **Enterprise Architect Object Model 2.10** checkbox from the list.
- Do the same for VB or VB Word: select the checkbox for the **Microsoft Word 10.0 Object Library**.
- Click on the **OK** button.

#### Note:

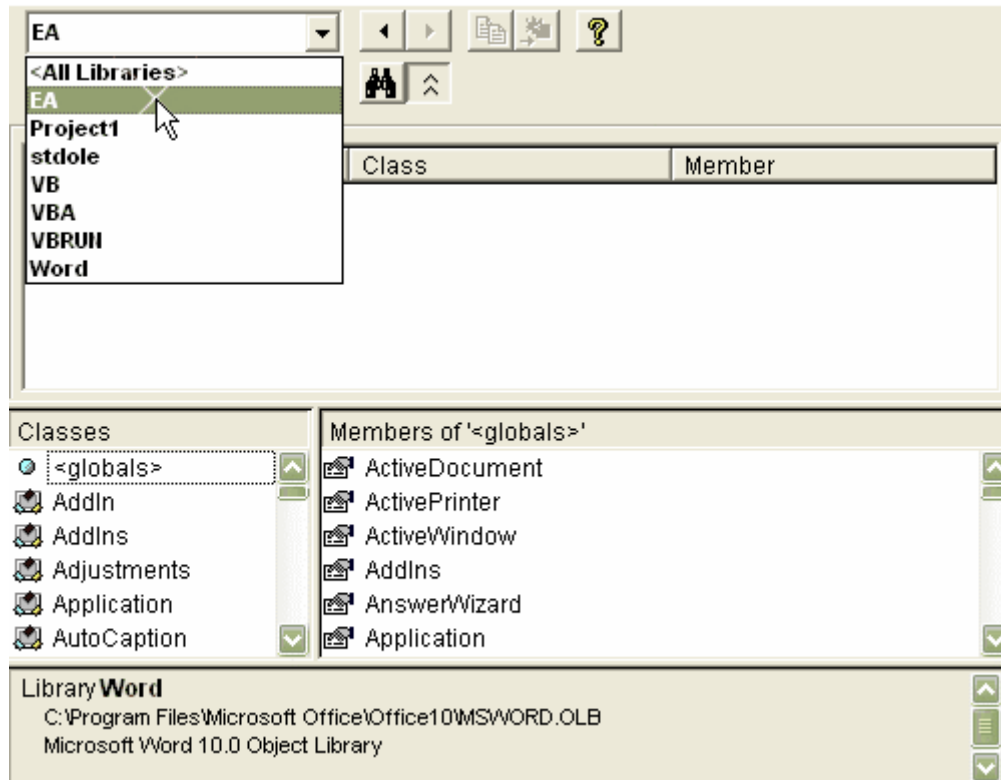
If Enterprise Architect Object Model 2.10 does not appear in the list, go to the command line and manually re-enter Enterprise Architect using the following:

- To unregister Enterprise Architect: `ea.exe /unregister`
- To register Enterprise Architect: `ea.exe /register`.

**Visual Basic 5/6 users** should also note that the version number of the Enterprise Architect interface is stored in the VBP project file in a form similar to the following:

Reference="G{64FB2BF4-9EFA-11D2-8307-C45586000000}#2.2#0#..\..\..\Program Files\Sparx Systems\EA\EA.TLB#Enterprise Architect Object Model 2.02

If you experience problems moving from one version of Enterprise Architect to another, open the VBP file in a text editor and remove this line. Then open the project in Visual Basic and use **Project-References** to create a new reference to the Enterprise Architect Object model.



Reference to objects in Enterprise Architect and Word should now be available in the **Object Browser**. This can be accessed from the main menu by selecting **View | Object Browser**, or by pressing **[F2]**.

The drop-down list on the top-left of the window should now include Enterprise Architect and Word. If MS-Project is installed this must also be set up.

## 7.1.2 Examples and Tips

Instructions for using the interface are provided through sample code. There are several sets of examples:

- VB 6 and C# examples are available in the *Code Samples* folder under your Enterprise Architect installation (default: C:\Program Files\Sparx Systems\EA\Code Samples)
- Enterprise Architect can be set up to [call an external application](#)<sup>[180]</sup>
- Several VB.NET code snippets are provided in the [reference section](#)<sup>[273]</sup>
- A comprehensive example of using Visual Basic to create MS Word documentation is available from the internet at [www.sparxsystems.com/resources/developers/autint\\_vb.html](http://www.sparxsystems.com/resources/developers/autint_vb.html)
- Additional samples are available from the Sparx Systems website; see the [Available Resources](#)<sup>[181]</sup> topic.

Additionally, you should note the following tips and tricks:

- An instance of the Enterprise Architect (EA.exe) process is executed when you initialize a new repository object. This process must remain running in order to perform automation tasks. If the main window is visible, you can safely minimize it, but it must remain running.
- The Enterprise Architect ActiveX Interface is a functional interface rather than a data interface. When you load data through the interface there is a noticeable delay as Enterprise Architect user interface elements (such as Windows, menus) are loaded and the specified database connection is established.
- Collections use a zero-based index; for example, Repository.Models(0) represents the first model in the

repository.

- During the development of your client software your program might terminate unexpectedly and leave EA.exe running in such a state that it is unable to support further interface calls. If your program terminates abnormally, ensure that Enterprise Architect is not left running in the background (see the Windows [Task Manager / Process](#) tab).
- A handle to a currently running instance of Enterprise Architect can be obtained through the use of a `GetObject()` call. For more information, refer to the reference page for the [App](#)<sup>[185]</sup> object. Accessing your Enterprise Architect model via the `App` object enables querying the current User Interface status, such as using `GetContextItem()` on the [Repository](#)<sup>[190]</sup> object to detect the current selection by the user, allowing for rapid prototyping and testing.

### Enterprise Architect Not Closing

If your automation controller was written using the .NET framework, Enterprise Architect does not close even after you release all your references to it. To force the release of the COM pointers, call the memory management functions as shown below:

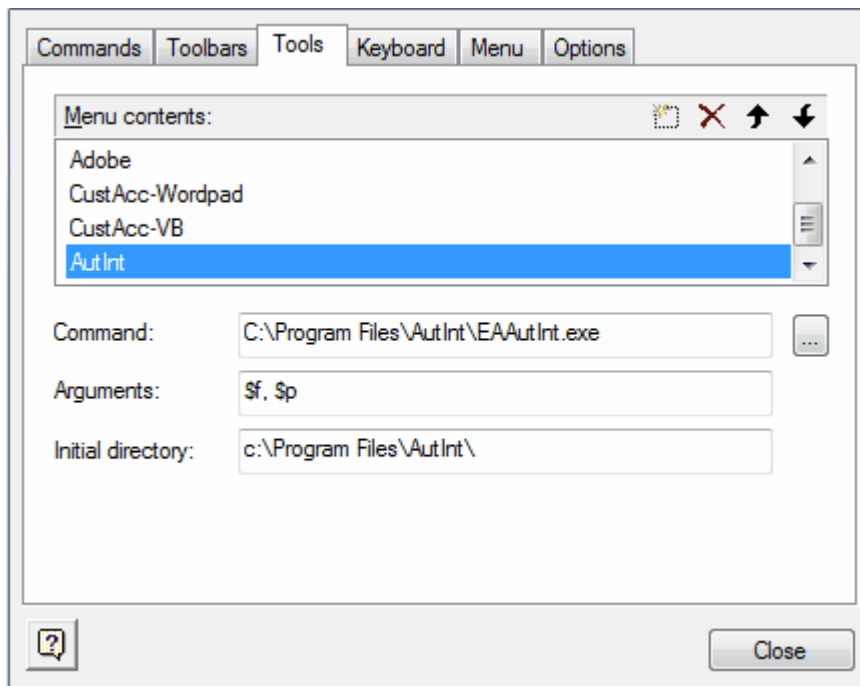
```
GC.Collect();
GC.WaitForPendingFinalizers();
```

There are additional concerns when controlling a running instance of Enterprise Architect that loads Add-Ins - see the [Tricks and Traps](#)<sup>[124]</sup> topic for details.

#### 7.1.2.1 Call from Enterprise Architect

Enterprise Architect can be set up to call an external application. You can pass parameters on the current position selected in the [Project Browser](#) to the application being called.

To define an application that you can run from Enterprise Architect, select the **Tools | Customize** menu option. The [Customize](#) dialog displays. Select the [Tools](#) tab.



With this you can:

- Add a command line for an application
- Define parameters to pass to this application

The parameters required for running the AutInt executable are:

- The Enterprise Architect file parameter `$f` and
- The current PackageID `$p`

Hence the arguments should simply contain: \$f, \$p

The available parameters for passing information to external applications are:

Parameter	Description	Notes
\$d	Diagram ID	ID for accessing associated diagram.
\$D	Diagram GUID	GUID for accessing the associated diagram.
\$e	Comma separated list of element IDs	All elements selected in the current diagram.
\$E	Comma separated list of element GUIDs	All elements selected in the current diagram.
\$f	Project Name	For example: C:\projects\EAexample.eap.
\$F	Calling Application (Enterprise Architect)	'Enterprise Architect'.
\$p	Current Package ID	For example: 144.
\$P	Package GUID	GUID for accessing this package.

Once this has been set up, the application can be called from the main menu in Enterprise Architect using the **Tools | YourApplication** menu option.

### 7.1.2.2 Available Resources

Other available resources include:

Resource	Download Link
VB 6 Add-In for generating MS Word documentation.	<a href="http://www.sparxsystems.com/resources/developers/autint_vb.html">www.sparxsystems.com/resources/developers/autint_vb.html</a>
VB 6 Add-In to display a custom ActiveX graph control within the Enterprise Architect window as a new view.	<a href="http://www.sparxsystems.com/resources/developers/autint_vb_custom_view.html">www.sparxsystems.com/resources/developers/autint_vb_custom_view.html</a>
A basic Add-In framework written in C#. Useful as a starting point for authoring your own custom Enterprise Architect Add-In.	<a href="http://www.sparxsystems.com/bin/CS_AddinFramework.zip">www.sparxsystems.com/bin/CS_AddinFramework.zip</a>
An extension on the <i>CS_AddinFramework</i> example showing how to export Tagged Values to a .csv file.	<a href="http://www.sparxsystems.com/bin/CS_AddinTaggedCSV.zip">www.sparxsystems.com/bin/CS_AddinTaggedCSV.zip</a>
A basic Add-In skeleton written in Delphi.	<a href="http://www.sparxsystems.com/bin/DelphiDemo.zip">www.sparxsystems.com/bin/DelphiDemo.zip</a>
A simple example Add-In written in C#.	<a href="http://www.sparxsystems.com/bin/CS_Sample.zip">www.sparxsystems.com/bin/CS_Sample.zip</a>

For further information, see [www.sparxsystems.com/resources/developers/autint.html](http://www.sparxsystems.com/resources/developers/autint.html).

## 7.2 Reference

This section provides detailed information on all the objects available in the object model provided by the Automation Interface, covering:

- [Interface Overview](#)<sup>[182]</sup>
- [App](#)<sup>[185]</sup>
- [Enumerations](#)<sup>[186]</sup>
- [Repository](#)<sup>[189]</sup>

- [Element](#) <sup>[218]</sup>
- [Element Features](#) <sup>[235]</sup>
- [Connector](#) <sup>[246]</sup>
- [Diagram Package](#) <sup>[254]</sup>
- [Project Interface](#) <sup>[262]</sup>
- [Code Samples](#) <sup>[273]</sup>

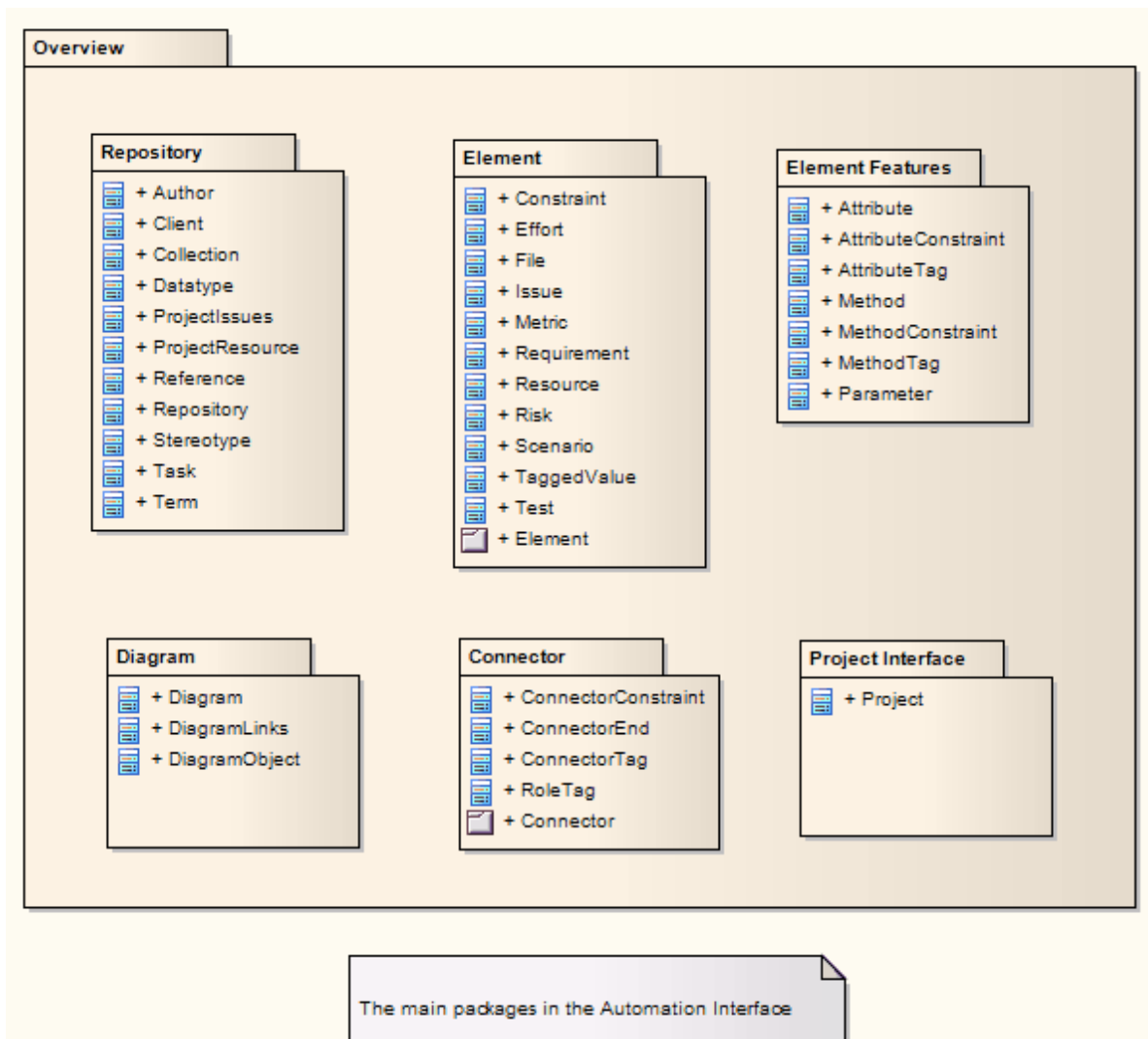
### 7.2.1 Interface Overview

#### public Package

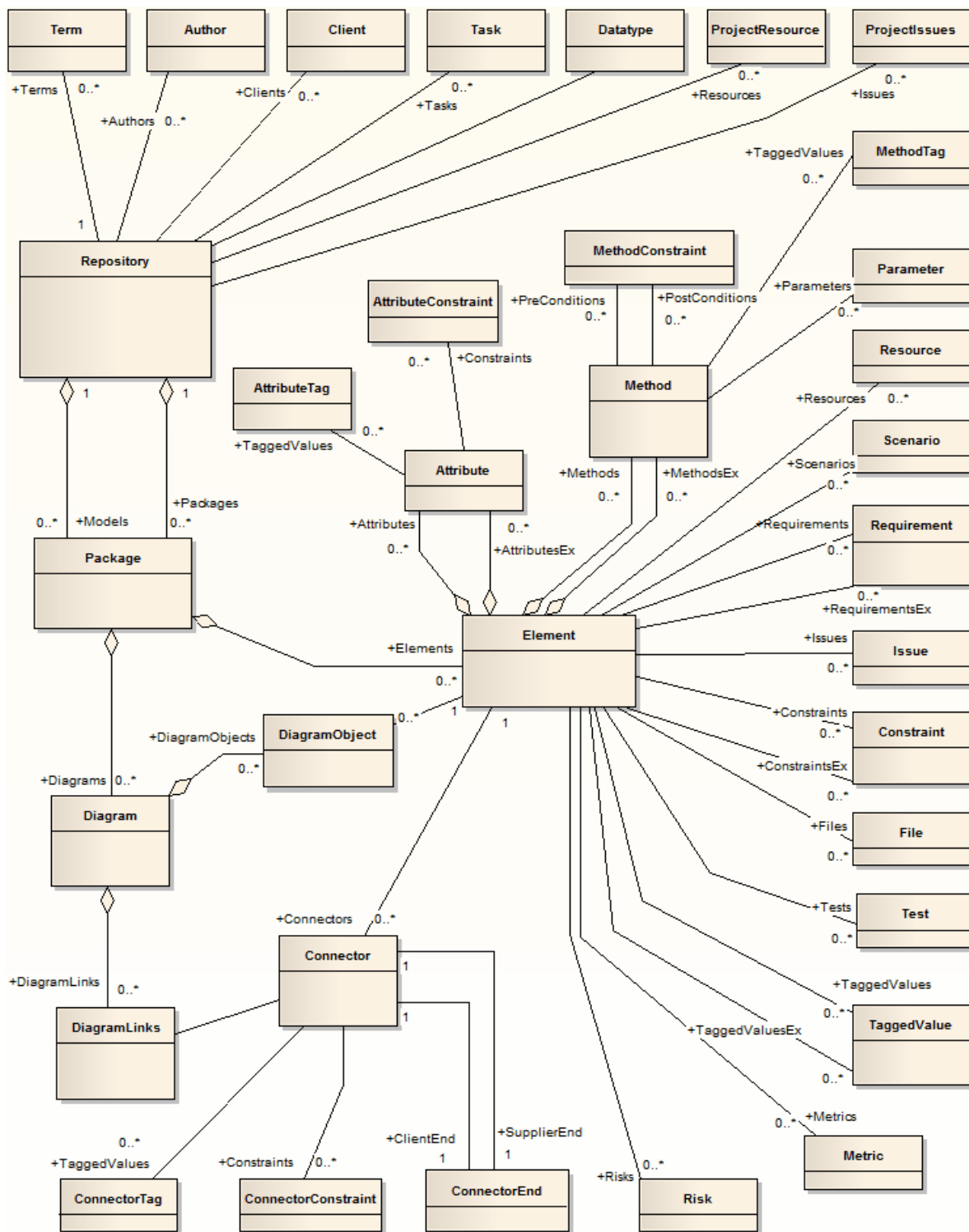
This package provides an overview of the main elements within the Automation Interface. These are:

- The [Repository](#) <sup>[189]</sup>, which represents the model as a whole and provides entry to model packages and collections
- [Elements](#) <sup>[218]</sup>, which are the basic structural unit (such as Class, Use Case and Object)
- [Element Features](#) <sup>[235]</sup>, which are attributes and operations defined on an element
- [Diagram Package](#) <sup>[254]</sup>, the visible drawings contained in the model
- [Connectors](#) <sup>[246]</sup>, relationships between elements.

The following diagram illustrates the main interface elements and their associated contents. Each element in this document is creatable by Automation and can be accessed through the various collections that exist or, in some cases, directly.



The following diagram provides a high level overview of the Automation Interface for accessing, manipulating, modifying and creating Enterprise Architect UML elements. The top level object is the Repository, which contains collections for a variety of system level objects, as well as the main Models collection that provides access to the UML elements, diagrams and packages within the project. In general, the Role names applied at the Target end of associations indicate the name of the Collection that is used to access instances of that object.



### Internal Links

- Logical diagram:: Automation Interface  
Package:: Automation Interface
- Logical diagram:: Automation Interface  
Package:: Automation Interface
- Logical diagram:: Automation Interface  
Package:: Automation Interface



- Logical diagram:: Automation Interface  
Package:: Automation Interface
- Logical diagram:: Automation Interface  
Package:: Automation Interface
- Logical diagram:: Automation Interface  
Package:: Automation Interface
- Logical diagram:: Automation Interface  
Package:: Automation Interface

## Connectors

Connector	Source	Target
<i>Nesting</i> source > target	<i>Connector</i> Contained Element	<i>Overview</i> Containing Element
<i>Nesting</i> source > target	<i>Repository</i> Contained Element	<i>Overview</i> Containing Element
<i>Nesting</i> source > target	<i>Diagram</i> Contained Element	<i>Overview</i> Containing Element
<i>Nesting</i> source > target	<i>Element</i> Contained Element	<i>Overview</i> Containing Element
<i>Nesting</i> source > target	<i>Project Interface</i> Contained Element	<i>Overview</i> Containing Element
<i>Nesting</i> source > target	<i>ElementFeatures</i> Contained Element	<i>Overview</i> Containing Element

## 7.2.2 App

The *App* object represents a running instance of Enterprise Architect. Its object provides access to the Automation Interface.

Attribute	Type	Notes
Project	Project	Read only. Provides a handle to the Project Interface.
Repository	Repository	Read only. Provides a handle to the Repository object.
Visible	Boolean	Read/Write. Whether or not the application is visible.

## GetObject() Support

The *App* object is creatable and a handle can be obtained by creating one. In addition, clients can use the equivalent of Visual Basic's *GetObject()* to obtain a reference to a currently running instance of Enterprise Architect.

Use this method to more quickly test changes to Add-Ins and external clients, as the Enterprise Architect application and data files do not have to be constantly re-loaded.

For example:

```
Dim App as EA.App
Set App = GetObject("EA.App")
MsgBox App.Repository.Models.Count
```

Another example, which uses the *App* object without saving it to a variable:

```
Dim Rep as EA.Repository
Set Rep = GetObject("EA.App").Repository
MsgBox Rep.ConnectionString
```

### 7.2.3 Enumerations

These enumerations are defined by the Automation Interface:

- [ConstLayoutStyles Enum](#) <sup>[186]</sup>
- [EnumRelationSetType Enum](#) <sup>[187]</sup>
- [MDGMenus Enum](#) <sup>[187]</sup>
- [ObjectType Enum](#) <sup>[188]</sup>
- [PropType Enum](#) <sup>[188]</sup>
- [ReloadType Enum](#) <sup>[189]</sup>
- [XMIType Enum](#) <sup>[189]</sup>

#### 7.2.3.1 ConstLayoutStyles Enum

The *enum* values defined here are used exclusively for the *Layout a Diagram* method. They enable you to define the layout options as depicted in the **Layout a Diagram** menu option. For further information, see the *Layout a Diagram* topic in *UML Modeling With Enterprise Architect - UML Modeling Tool*.

Method	Use to
<i>IsCrossReduceAggressive</i>	Perform aggressive Cross-reduction in the layout process (time consuming).
<i>IsCycleRemoveDFS</i>	Use the <i>Depth First Cycle Removal</i> algorithm.
<i>IsCycleRemoveGreedy</i>	Use the <i>Greedy Cycle Removal</i> algorithm.
<i>IsDiagramDefault</i>	Use existing layout options specified for this diagram.
<i>IsInitializeDFSIn</i>	Initialize the layout using the <i>Depth First Search Inward</i> algorithm.
<i>IsInitializeNaive</i>	Initialize the layout using the <i>Naïve Initialize Indices</i> algorithm.
<i>IsInitializeDFSOut</i>	Initialize the layout using the <i>Depth First Search Outward</i> algorithm.
<i>IsLayeringLongestPathSink</i>	Layer the diagram using the <i>Longest Path Sink</i> algorithm.
<i>IsLayeringLongestPathSource</i>	Layer the diagram using the <i>Longest Path Source</i> algorithm.
<i>IsLayeringOptimalLinkLength</i>	Layer the diagram using the <i>Optimal Link Length</i> algorithm.
<i>IsLayoutDirectionDown</i>	Direct connectors to point downwards.
<i>IsLayoutDirectionLeft</i>	Direct connectors to point leftwards.
<i>IsLayoutDirectionRight</i>	Direct connectors to point rightwards.
<i>IsLayoutDirectionUp</i>	Direct connectors to point upwards.
<i>IsProgramDefault</i>	Use factory default layout options as specified by Enterprise Architect.

#### 7.2.3.2 CreateModelType Enum

The *CreateModelType* enumeration is used for the [CreateModel](#) <sup>[194]</sup> method on the Repository Class.

Method	Use to
<i>cmEAPFromBase</i>	Create a copy of the EABase model file to the specified file path.
<i>cmEAPFromSQLRepository</i>	Create a .eap file shortcut to an SQL-based repository. Requires user interaction to provide SQL connection details.

### 7.2.3.3 *EAEditionTypes Enum*

The *EAEditionTypes* enumeration identifies the level of licensed functionality available to the current repository. For example:

```
EAEditionTypes theEdition = theRepository.GetEAEdition();
if ( theEdition == EAEditionTypes.piDesktop )
    ...
else if ( theEdition == EAEditionTypes.piProfessional )
    ...
```

The enumeration defines the following formal values:

- *piLite*
- *piDesktop*
- *piProfessional*
- *piCorporate*
- *piBusiness*
- *piSystemEng*
- *piUltimate*.

There is no separate value for the trial edition; the *Repository.EAEdition* attribute contains the appropriate **EAEditionTypes** value for whichever edition the user has selected to trial.

### 7.2.3.4 *EnumRelationSetType Enum*

This enumeration represents values returned from the *GetRelationSet* method of the [Element](#)<sup>[22†]</sup> object.

Method	Notes
<i>rsDependEnd</i>	List of elements that depend on the current element.
<i>rsDependStart</i>	List of elements that the current element depends on.
<i>rsGeneralizeEnd</i>	List of elements that are generalized by the current element.
<i>rsGeneralizeStart</i>	List of elements that the current element generalizes.
<i>rsParents</i>	List of all parent elements of the current element.
<i>rsRealizeEnd</i>	List of elements that are realized by the current element.
<i>rsRealizeStart</i>	List of elements that the current element realizes.

### 7.2.3.5 *MDGMenus Enum*

Use this enumeration when providing the *HiddenMenus* property to *MDG\_GetProperty*.

These options are exclusive of one another and can be read or added to hide more than one menu.

See the [MDG\\_GetProperty](#)<sup>[168†]</sup> topic for an example of use.

Method	Use to
<i>mgBuildProject</i>	Hide <b>Build Project</b> menu option.
<i>mgMerge</i>	Hide <b>Merge</b> menu option.
<i>mgRun</i>	Hide <b>Run</b> menu option.

### 7.2.3.6 ObjectType Enum

The *ObjectType* enumeration identifies Enterprise Architect object types even when referenced through a Dispatch interface.

For example:

```
Object ob = Repository.GetElementByID(13);
if ( ob.ObjectType == otElement )
;
else if( ob.ObjectType == otAuthor )
...

```

All of the following are valid enumeration values:

<i>otNone</i>	<i>otClient</i>
<i>otProject</i>	<i>otAuthor</i>
<i>otRepository</i>	<i>otDatatype</i>
<i>otCollection</i>	<i>otStereotype</i>
<i>otElement</i>	<i>otTaskotTerm</i>
<i>otPackage</i>	<i>otProjectIssues</i>
<i>otModel</i>	<i>otAttributeConstraint</i>
<i>otConnector</i>	<i>otAttributeTag</i>
<i>otDiagram</i>	<i>otMethodConstraint</i>
<i>otRequirement</i>	<i>otMethodTag</i>
<i>otScenario</i>	<i>otConnectorConstrain</i>
<i>otConstraint</i>	<i>otConnectorTag</i>
<i>otTaggedValue</i>	<i>otProjectResource</i>
<i>otFile</i>	<i>otReference</i>
<i>otEffort</i>	<i>otRoleTag</i>
<i>otMetric</i>	<i>otCustomProperty</i>
<i>otIssue</i>	<i>otPartition</i>
<i>otRisk</i>	<i>otTransition</i>
<i>otTest</i>	<i>otEventProperty</i>
<i>otDiagramObject</i>	<i>otEventProperties</i>
<i>otDiagramLink</i>	<i>otPropertyType</i>
<i>otResource</i>	<i>otProperties</i>
<i>otConnectorEnd</i>	<i>otProperty</i>
<i>otAttribute</i>	<i>otSwimlaneDef</i>
<i>otMethod</i>	<i>otSwimlanes</i>
<i>otParameter</i>	<i>otSwimlane</i>

### 7.2.3.7 PropType Enum

The *PropType* enumeration gives the automation programmer an indication of what sort of data is going to be stored by this property.

Method	Notes
<i>ptArray</i>	An array containing values of any type.
<i>ptBoolean</i>	True or False.
<i>ptEnum</i>	A string being an entry in the semi-colon separated list specified in the validation field of the Property.
<i>ptFloatingPoint</i>	4 or 8 byte floating point value.
<i>ptInteger</i>	16-bit or 32-bit signed integer.
<i>ptString</i>	Unicode string.

### 7.2.3.8 ReloadType Enum

This enumeration represents values returned from the *GetReloadItem* and *PeekReloadItem* methods of the *ModelWatcher* Class. It has four possible values, which define the type of change that was made to a model.

Method	Notes
<i>rtElement</i>	The <i>Item</i> parameter represents a particular element that must be reloaded.
<i>rtEntireModel</i>	Entire model must be reloaded to ensure that all changes are reloaded.
<i>rtNone</i>	No change in the model.
<i>rtPackage</i>	The <i>Item</i> parameter represents a particular package that must be reloaded.

### 7.2.3.9 XMISpec Enum

The following enumeration values are used in the *Project.ExportPackageXML()* method. They enable specification of the XML export type.

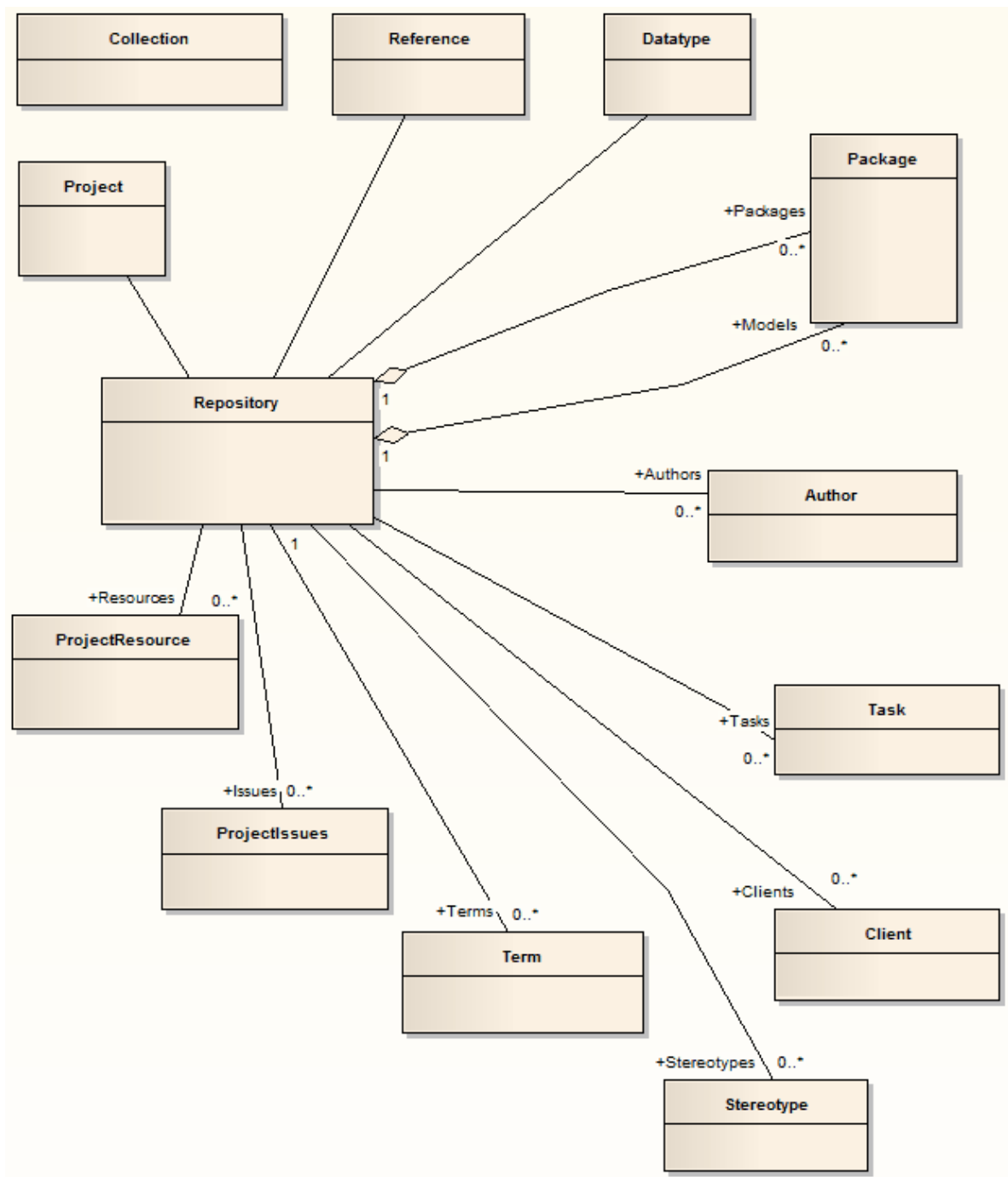
*xmiEADefault*  
*xmiRoseDefault*  
*xmiEA10*  
*xmiEA11*  
*xmiEA12*  
*xmiRose10*  
*xmiRose11*  
*xmiRose12*  
*xmiMOF13*  
*xmiMOF14*  
*xmiEA20*  
*xmiEA21*

## 7.2.4 Repository

### public Package

The *Repository* package contains the high level system objects and entry point into the model itself using the *Models* collection and the other system level collections.

This diagram illustrates the [Repository](#)<sup>[190]</sup> and its first level functions and collections.



### 7.2.4.1 Repository

#### public Class

The *Repository* is the main container of all structures such as models, packages and elements. You can iteratively begin accessing the model using the *Models* collection. It also has some convenient methods to directly access the structures without having to locate them in the hierarchy first.

Associated table in .EAP file: *<none>*

## Repository Attributes

Attribute	Type	Notes
<b>Authors</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. The system <i>Authors</i> collection. Contains 0 or more <i>Author objects</i> , each of which can be associated with, for example, elements or diagrams as the item author or owner. Use <i>AddNew</i> , <i>Delete</i> and <i>GetAt</i> to manage Authors.
<b>BatchAppend</b>	<i>Boolean</i>	Read/Write. Set this property to <b>true</b> when your automation client has to rapidly insert many elements, operations, attributes and/or operation parameters. Set to <b>false</b> when work is complete.  This can result in 10- to 20-fold improvement in adding new elements in bulk.
<b>Clients</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. A list of <i>Clients</i> associated with the project. You can modify, delete and add new <i>Client objects</i> using this collection.
<b>ConnectionString</b>	<i>String</i>	Read only. The filename/connection string of the current Repository.
<b>Datatypes</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. The <i>Datatypes</i> collection. Contains a list of <i>Datatype objects</i> , each representing a data type definition for either data modeling or code generation purposes.
<b>EAEdition</b>	<a href="#">EAEditionTypes</a> <sup>[187]</sup>	Read only. Returns the level of core licensed functionality available to the current repository.  <b>Note:</b>  This property returns <b>Corporate</b> when the edition is <i>Business and Software Engineering</i> , <i>Systems Engineering</i> or <i>Ultimate</i> . Use <b>EAEditionEx</b> to identify which of these extended editions is available.
<b>EAEditionEx</b>	<a href="#">EAEditionTypes</a> <sup>[187]</sup>	Read only. Returns the level of extended licensed functionality available to the current repository.
<b>EnableCache</b>	<i>Boolean</i>	Read/Write. An optimization for pre-loading package objects when dealing with large sets of automation objects.
<b>EnableUIUpdates</b>	<i>Boolean</i>	Read/Write. Set this property to <b>false</b> to improve the performance of changes to the model; for example, bulk addition of elements to a package. To reveal the changes to the user, call <i>Repository.RefreshModel/View()</i> .
<b>FlagUpdate</b>	<i>Boolean</i>	Read/Write. Instructs Enterprise Architect to update the Repository with the <i>LastUpdate</i> value.
<b>InstanceGUID</b>	<i>String</i>	Read only. The identifier string identifying the Enterprise Architect runtime session.
<b>IsSecurityEnabled</b>	<i>Boolean</i>	Read only. Checks whether User Security is enabled for the current repository.
<b>Issues</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. The <i>System Issues</i> list. Contains <i>ProjectIssues objects</i> , each detailing a particular issue as it relates to the project as a whole.

Attribute	Type	Notes
LastUpdate	String	Read only. The identifier string identifying the Enterprise Architect runtime session and the timestamp for when it was set.
LibraryVersion	Long	Read only. The build number of the Enterprise Architect runtime.
Models	<a href="#">Collection</a> <sup>[205]</sup> of type <a href="#">Package</a> <sup>[209]</sup>	<p>Read only. <i>Models</i> are of type <i>package</i> and belong to a collection of packages. This is the top level entry point to an Enterprise Architect project file. Each model is a <i>root node</i> in the <b>Project Browser</b> and can contain items such as Views and packages.</p> <p>A model is a special form of a package; it has a <i>ParentID</i> of <b>0</b>. By iterating through all models, you can access all the elements within the project hierarchy.</p> <p>You can also use the <i>AddNew</i> function to create a new model. A model can be deleted, but remember that everything contained in the model is deleted as well.</p>
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through the Dispatch interface.
ProjectGUID	String	Read only. Returns a unique ID for the project.
PropertyTypes	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of <a href="#">Property Types</a> <sup>[214]</sup> available to the Repository.
Resources	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Contains available <i>ProjectResource</i> objects to assign to work items within the project. Use the <b>add new</b> , <b>modify</b> and <b>delete</b> functions to manage resources.
Stereotypes	<a href="#">Collection</a> <sup>[205]</sup>	Read only. The <a href="#">Stereotypes</a> <sup>[216]</sup> collection. A list of <i>Stereotype objects</i> that contain information on a stereotype and which elements it can be applied to.
SuppressEADialogs	Boolean	Read/Write. Set this property in the <a href="#">EA_OnPostNewElement</a> <sup>[142]</sup> or <a href="#">EA_OnPostNewConnector</a> <sup>[142]</sup> broadcast events to control whether Enterprise Architect should suppress showing the default <b>Properties</b> dialogs to the user when an element or connector is created.
Tasks	<a href="#">Collection</a> <sup>[205]</sup>	Read only. A list of system tasks (to do list). Each entry is a <a href="#">Task</a> <sup>[216]</sup> <i>Item</i> ; you can modify, delete and add new tasks.
Terms	<a href="#">Collection</a> <sup>[205]</sup>	Read only. The project <i>Glossary</i> . Each <a href="#">Term</a> <sup>[217]</sup> object is an entry in the Glossary. Add, modify and delete Terms to maintain the Glossary.

### Repository Methods

Method	Type	Notes
ActivateDiagram (long DiagramID)		<p>Activates an already open diagram (that is, makes it the active tab) in the main Enterprise Architect user interface.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>DiagramID: Long - the ID of the diagram to make active.</li> </ul>



Method	Type	Notes
ActivatePerspective (string, long)	Boolean	<b>Deprecated</b> - no longer in use.
ActivateTab (string Name)		Activates an open Enterprise Architect tabbed view. Parameters: <ul style="list-style-type: none"> <li>Name: String - the name of the view to activate.</li> </ul>
ActivateTechnology (string Name)		Activates an enabled MDG Technology. Parameters: <ul style="list-style-type: none"> <li>Name: String - the name of the Technology to activate.</li> </ul>
ActivateToolbox (string Toolbox, long Options)	Boolean	Activates a <b>Toolbox</b> page in the GUI. Returns <b>true</b> if the specified <b>Toolbox</b> page is successfully activated, otherwise returns <b>false</b> . Parameters: <ul style="list-style-type: none"> <li>Toolbox: String - the name of the <b>Toolbox</b> page to activate.</li> <li>Options: Long - reserved for future use.</li> </ul>
AddDefinedSearches (string sXML)		Enables you to enter a set of defined searches that last in Enterprise Architect for the life of the application. When Enterprise Architect loads again they must be inserted again by your Add-In. Parameters: <ul style="list-style-type: none"> <li>sXML: String - the XML of the defined searches; you can get this XML by performing an <i>export</i> of the searches from the <b>Manage Searches</b> dialog in Enterprise Architect. (See <i>Using Enterprise Architect - UML Modeling Tool</i>.)</li> </ul>
AddPerspective (string Perspective, long Options)	Boolean	<b>Deprecated</b> - no longer in use.
AddTab (string TabName, string ControlID)	activeX custom control	Adds an ActiveX custom control as a tabbed window. Enterprise Architect creates a control and, if successful, returns its Unknown pointer, which can be used by the caller to manipulate the control. Parameters: <ul style="list-style-type: none"> <li>TabName: String - used as the tab caption.</li> <li>ControlID: String - the ProgID of the control. for example, Project1, UserControl1.</li> </ul>
AdviseConnectorChange (long ConnectorID)		Provides an Add-In or automation client with the ability to advise the Enterprise Architect user interface that a particular connector has changed and, if it is visible in any open diagram, to reload and refresh that connector for the user. Parameters: <ul style="list-style-type: none"> <li>ConnectorID: Long - the ID of the connector.</li> </ul>
AdviseElementChange (long ObjectID)		Provides an Add-In or automation client with the ability to advise the Enterprise Architect user interface that a particular element has changed and, if it is visible in any open diagram, to reload and refresh that element for the user.

Method	Type	Notes
		Parameters: <ul style="list-style-type: none"> <li>ObjectID: Long - the ID of the element.</li> </ul>
<b>ChangeLoginUser (string Name, string Password)</b>	<i>Boolean</i>	Sets the currently logged on user to be that specified by a name and password. This logs the user into the repository when security is enabled. If security is not enabled an exception ( <i>Security not enabled</i> ) is thrown.  Parameters: <ul style="list-style-type: none"> <li>Name: String - the name of the user.</li> <li>Password: String - the password of the user.</li> </ul>
<b>ClearAuditLogs (Object StartDateTime, Object EndDateTime)</b>	<i>Boolean</i>	Clears all Audit Logs from the model. If <i>StartDateTime</i> and <i>EndDateTime</i> are not null then only log items that fall into this period are cleared. Returns <b>true</b> for success, <b>false</b> for failure.  <b>Notes:</b> <ul style="list-style-type: none"> <li>This method cannot be undone. It is strongly advised that you call <i>SaveAuditLogs</i> first to backup the logs.</li> <li>This method might fail if the user logged into the model does not have the correct access permission.</li> </ul> Parameters: <ul style="list-style-type: none"> <li>StartDateTime: Variant [DateTime] - the earliest date and time of log entries to clear.</li> <li>EndDateTime; Variant [DateTime] - the latest date and time of log entries to clear.</li> </ul>
<b>ClearOutput (string Name)</b>		Removes all the text from a tab in the <b>Output</b> window. See also <a href="#">CreateOutputTab</a> <sup>[195]</sup> , <a href="#">EnsureOutput Visible</a> <sup>[195]</sup> , <a href="#">WriteOutput</a> <sup>[203]</sup> .  Parameters: <ul style="list-style-type: none"> <li>Name: String - the name of the tab to remove text from.</li> </ul>
<b>CloseAddins ()</b>		Called by automation controllers to ensure that Add-Ins created in .NET do not linger after all controller references to Enterprise Architect have been cleared.
<b>CloseDiagram (long DiagramID)</b>		Closes a diagram in the current list of diagrams that Enterprise Architect has open.  Parameters: <ul style="list-style-type: none"> <li>DiagramID: Long - the ID of the diagram to close.</li> </ul>
<b>CloseFile ()</b>		Closes any open file.
<b>CreateModel (CreateModelType CreateType, string FilePath, long ParentWnd)</b>	<i>Boolean</i>	Creates a new .eap model file based on the standard Enterprise Architect Base model, or a shortcut .eap based on a provided SQL connection.  Returns <b>true</b> when the new file is created, otherwise returns <b>false</b> .  Parameters: <ul style="list-style-type: none"> <li>CreateType: <a href="#">CreateModelType</a><sup>[186]</sup> - Specify whether to make a new copy of the EABase.eap model, or</li> </ul>

Method	Type	Notes
		<p>create a .eap file shortcut to a DBMS repository. The latter option requires a dialog to be opened for the user to provide SQL connection details.</p> <ul style="list-style-type: none"> <li>• FilePath: String - Destination for new .eap file.</li> <li>• ParentWnd: Long - Window handle to act as the parent for the SQL connection dialog. Only required when using <i>cmEAPFromSQLRepository</i>.</li> </ul>
CreateOutputTab (string Name)		<p>Creates a tab in the <b>Output</b> window. See also <a href="#">ClearOutput</a><sup>[194]</sup>, <a href="#">EnsureOutput Visible</a><sup>[195]</sup>, <a href="#">WriteOutput</a><sup>[203]</sup>.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Name: String - the name of the tab to create.</li> </ul>
DeletePerspective (string Perspective, long Options)	Boolean	<b>Deprecated</b> - no longer in use.
DeleteTechnology (string ID)	Boolean	<p>Removes a specified MDG Technology resource from the repository.</p> <p>Returns <b>true</b>, if the technology is successfully removed from the model. Returns <b>false</b> otherwise.</p> <p><b>Note:</b></p> <p>This applies to technologies imported into pre-7.0 versions of Enterprise Architect (imported technologies), not to technologies referenced in version 7.0 and later (referenced technologies). See <a href="#">Deploying MDG Technologies</a><sup>[58]</sup> (from Add-Ins).</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ID: String - the ID of the technology.</li> </ul>
EnsureOutputVisible (string Name)		<p>Ensures that a specified tab in the <b>Output</b> window is visible to the user. The <b>Output</b> window is made visible if it is hidden. See also <a href="#">ClearOutput</a><sup>[194]</sup>, <a href="#">CreateOutputTab</a><sup>[195]</sup>, <a href="#">WriteOutput</a><sup>[203]</sup>.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Name: String - the name of the tab to make visible.</li> </ul>
ExecutePackageBuildScript (long ScriptOptions, string PackageGuid)		<p>Enables you to run the active package build script based on your current selection in the <b>Project Browser</b>. You can also run a script by passing in the package GUID.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• ScriptOptions: Long - the script type; can be any one of these numerical values: <ul style="list-style-type: none"> <li>1 = Build</li> <li>2 = Test</li> <li>3 = Run</li> <li>4 = Create Workbench Instance</li> <li>5 = Debug.</li> </ul> </li> <li>• PackageGuid: String - the ID of the package for which to run the script.</li> </ul>
Exit		Shuts down Enterprise Architect immediately. Used by DotNET programmers where the garbage collector does not immediately release all referenced COM objects.

Method	Type	Notes
<b>GetActivePerspective ()</b>	<i>String</i>	<b>Deprecated</b> - no longer in use.
<b>GetAttributeByGuid (string Guid)</b>	<a href="#">Attribute</a> <sup>[235]</sup>	Returns a pointer to an attribute in the repository, located by its GUID. This is usually found using the <i>AttributeGUID</i> property of an attribute.  Parameters: <ul style="list-style-type: none"> <li>Guid: String - the GUID of the attribute to locate.</li> </ul>
<b>GetAttributeByID (string Id)</b>	<a href="#">Attribute</a> <sup>[235]</sup>	Returns a pointer to an attribute in the repository, located by its ID. This is usually found using the <i>AttributeID</i> property of an attribute.  Parameters: <ul style="list-style-type: none"> <li>Id: String - the ID of the attribute to locate.</li> </ul>
<b>GetConnectorByGuid (string Guid)</b>	<a href="#">Connector</a> <sup>[248]</sup>	Returns a pointer to a connector in the repository, located by its GUID. This is usually found using the <i>ConnectorGUID</i> property of a connector.  Parameters: <ul style="list-style-type: none"> <li>Guid: String - the GUID of the connector to locate.</li> </ul>
<b>GetConnectorByID (long ConnectorID)</b>	<a href="#">Connector</a> <sup>[248]</sup>	Searches the repository for a connector with a specific ID.  Parameters: <ul style="list-style-type: none"> <li>ConnectorID: Long - the ID of the connector to locate.</li> </ul>
<b>GetContextItem (object Item)</b>	<a href="#">ObjectType</a> <sup>[188]</sup>	Sets a pointer to an item in context within Enterprise Architect.  Also returns the corresponding <i>ObjectType</i> .  For additional information about <i>ContextItems</i> and the supported <i>ObjectTypes</i> see the <b>GetContextItemType</b> method (below).  Parameters: <ul style="list-style-type: none"> <li>Item: Object - the item to point to.</li> </ul>
<b>GetContextItemType ()</b>	<a href="#">ObjectType</a> <sup>[188]</sup>	Returns the <i>ObjectType</i> of an item in context within Enterprise Architect. A <i>ContextItem</i> is defined as an item selected anywhere within the Enterprise Architect GUI including: <ul style="list-style-type: none"> <li>An item selected in the <b>Project Browser</b></li> <li>An item selected in an open diagram</li> <li>An item selected in certain dialogs, such as the attribute <b>Properties</b> dialog.</li> </ul> The supported <i>ObjectTypes</i> can be any one of the following values: <ul style="list-style-type: none"> <li><i>otElement</i></li> <li><i>otPackage</i></li> <li><i>otDiagram</i></li> <li><i>otAttribute</i></li> <li><i>otMethod</i></li> <li><i>otConnector</i></li> </ul>
<b>GetCurrentContextObject ()</b>	<i>Object</i>	Returns the current context Object.

Method	Type	Notes
<b>GetCounts ()</b>	<i>String</i>	Returns a set of counts from a number of tables within the base Enterprise Architect repository. These can be used to determine whether records have been added or deleted from the tables for which information is retrieved.
<b>GetCurrentDiagram ()</b>	<a href="#">Diagram</a> <sup>[255]</sup>	Returns a selected diagram.
<b>GetCurrentLoginUser (boolean GetGuid = false)</b>	<i>String</i>	If security is not enabled in the repository, an error is generated.  If <i>GetGuid</i> is <b>True</b> , a GUID generated by Enterprise Architect representing the user is returned; otherwise the text as entered in <i>System Users/User Details/Login</i> is returned.
<b>GetDiagramByGuid (string Guid)</b>	<a href="#">Diagram</a> <sup>[255]</sup>	Returns a pointer to a diagram using the global reference ID (global ID). This is usually found using the diagram <i>GUID</i> property of an element, and stored for later use to open a diagram without using the collection <i>GetAt()</i> function.  Parameters: <ul style="list-style-type: none"> <li>• Guid: String - the GUID of the diagram to locate.</li> </ul>
<b>GetDiagramByID (long DiagramID)</b>	<a href="#">Diagram</a> <sup>[255]</sup>	Gets a pointer to a diagram using an absolute reference number (local ID). This is usually found using the <i>DiagramID</i> property of an element, and stored for later use to open a diagram without using the collection <i>GetAt()</i> function.  Parameters: <ul style="list-style-type: none"> <li>• DiagramID: Long - the ID of the diagram to locate.</li> </ul>
<b>GetElementByGuid (string Guid)</b>	<a href="#">Element</a> <sup>[221]</sup>	Returns a pointer to an element in the repository, using the element's GUID reference number (global ID). This is usually found using the <i>ElementGUID</i> property of an element, and stored for later use to open an element without using the collection <i>GetAt()</i> function.  Parameters: <ul style="list-style-type: none"> <li>• Guid: String - the GUID of the element to locate.</li> </ul>
<b>GetElementByID (long ElementID)</b>	<a href="#">Element</a> <sup>[221]</sup>	Gets a pointer to an element using an absolute reference number (local ID). This is usually found using the <i>ElementID</i> property of an element, and stored for later use to open an element without using the collection <i>GetAt()</i> function.  Parameters: <ul style="list-style-type: none"> <li>• ElementID: Long - the ID of the element to locate.</li> </ul>
<b>GetElementsByQuery (string QueryName, string SearchTerm)</b>		Enables the user to run a search in Enterprise Architect, returning the result as a collection.  For example <i>GetElementsByQuery('Simple', 'Class1')</i> , where results contain elements with <i>Class1</i> in the <b>Name</b> and <b>Notes</b> fields.  Parameters: <ul style="list-style-type: none"> <li>• QueryName: String - the name of the search to run, for example 'Simple'.</li> <li>• SearchTerm: String - the term to search for.</li> </ul>
<b>GetElementSet (string</b>	<a href="#">Collection</a> <sup>[205]</sup>	Returns a set of elements as a collection based on a

Method	Type	Notes
IDList, long Options)		<p>comma-separated list of <i>ElementID</i> values. By default, if no values are provided in the IDList parameter, all objects for the entire project are returned.</p> <p>Parameters</p> <ul style="list-style-type: none"> <li>• IDList: String - a comma-separated list of <i>ElementID</i> values</li> <li>• Options: Long - modifies default behaviour of this method <ul style="list-style-type: none"> <li>• 1 - Returns empty collection when empty IDList parameter is given.</li> <li>• 2 - Use IDList string as an SQL query to populate this collection.</li> </ul> </li> </ul>
GetFieldFromFormat (string Format, string Text)	String	<p>Converts a field from your preferred format to Enterprise Architect's internal format. Returns the field in Enterprise Architect's internal format.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Format: String - The format to convert the field from. Valid formats are: <ul style="list-style-type: none"> <li>• HTML - Full HTML</li> <li>• RTF - Rich Text Format</li> <li>• TXT - Plain text</li> </ul> </li> <li>• Text: String - The field to be converted.</li> </ul>
GetFormatFromField (string Format, string Text)	String	<p>After accessing a field that contains formatting, use this method to convert it to your preferred format. Returns the field in the format specified.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Format: String - The format to convert the field to. Valid formats are: <ul style="list-style-type: none"> <li>• HTML - Full HTML</li> <li>• RTF - Rich Text Format</li> <li>• TXT - Plain text</li> </ul> </li> <li>• Text: String - The field to be converted.</li> </ul>
GetLastError ()	String	<p>Returns a string value describing the most recent error that occurred in relation to this object.</p> <p>This function is rarely used, as an exception is thrown when an error occurs.</p>
GetMethodByGuid (string Guid)	<a href="#">Method</a> <sup>[240]</sup>	<p>Returns a pointer to a method in the repository. This is usually found using the <i>MethodGUID</i> property of a method.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Guid: String - the GUID of the method to look for.</li> </ul>
GetMethodById (string Id)	<a href="#">Method</a> <sup>[240]</sup>	<p>Returns a pointer to a method in the repository. This is usually found using the <i>MethodID</i> property of a method.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• Id: String - the ID of the method to look for.</li> </ul>
GetPackageByGuid (string Guid)	<a href="#">Package</a> <sup>[209]</sup>	<p>Returns a pointer to a package in the repository using the package's GUID reference number (global ID). This is usually found using the <i>PackageGUID</i> property of the package.</p>

Method	Type	Notes
		<p>Each package in the model also has an associated element with the same GUID, so if you have an element with <i>Type="Package"</i> then you can load the package by calling:</p> <p><i>GetPackageByGuid(Element.ElementGUID)</i>.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>Guid: String - the GUID of the package to look for.</li> </ul>
<b>GetPackageByID (long PackageID)</b>	<a href="#">Package</a> <sup>[209]</sup>	<p>Get a pointer to a package using an absolute reference number (local ID). This is usually found using the <i>PackageID</i> property of an package, and stored for later use to open a package without using the collection <i>GetAt ()</i> function.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageID: Long - the ID of the package to locate.</li> </ul>
<b>GetProjectInterface ()</b>	<a href="#">Project</a> <sup>[262]</sup>	<p>Return a pointer to the <a href="#">EA.Project interface</a><sup>[262]</sup> (the XML-based automation server for Enterprise Architect). Use this interface to work with Enterprise Architect using XML, and also to access utility functions for loading diagrams, running reports and so on.</p>
<b>GetReferenceList (string Type)</b>	<a href="#">Reference</a> <sup>[215]</sup>	<p>Uses the list type to get a pointer to a <i>Reference List</i> object.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>Type: String - specifies the list type to get; valid list types are:</li> </ul> <p><i>Diagram</i>  <i>Element</i>  <i>Constraint</i>  <i>Requirement</i>  <i>Connector</i>  <i>Status</i>  <i>Cardinality</i>  <i>Effort</i>  <i>Metric</i>  <i>Scenario</i>  <i>Status</i> and  <i>Test</i>.</p>
<b>GetTechnologyVersion (string ID)</b>	String	<p>Returns the version of a specified MDG Technology resource.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>ID: String - the specified technology ID.</li> </ul>
<b>GetTreeSelectedItem (object SelectedItem)</b>	<a href="#">ObjectType</a> <sup>[188]</sup>	<p>Gets an object variable and type corresponding to the currently selected item in the tree view.</p> <p>To use this function, create a generic object variable and pass this as the parameter. Depending on the return type, cast it to a more specific type.</p> <p>The object passed back through the parameter can be a package, element, diagram, attribute or operation object.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>SelectedItem: Object - the object to get the variable and type for.</li> </ul>

Method	Type	Notes
<b>GetTreeSelectedItemType ()</b>	<a href="#">ObjectType</a> <sup>[188]</sup>	Returns the type of the object currently selected in the tree. One of: <ul style="list-style-type: none"> <li>• <i>otDiagram</i></li> <li>• <i>otElement</i></li> <li>• <i>otPackage</i></li> <li>• <i>otAttribute</i></li> <li>• <i>otMethod</i>.</li> </ul>
<b>GetTreeSelectedObject ()</b>	<i>Object</i>	The related method <a href="#">GetTreeSelectedItem</a> <sup>[199]</sup> () has an output parameter that is inaccessible by some scripting languages. As an alternative, this method provides the selected item through the return value.
<b>GetTreeSelectedPackage ()</b>	<a href="#">Package</a> <sup>[209]</sup>	Returns the package in which the currently selected tree view object is contained.
<b>HasPerspective (string Perspective)</b>	<i>String</i>	<b>Deprecated</b> - no longer in use.
<b>ImportPackageBuildScripts (string PackageGuid, string BuildScriptXML)</b>		Imports build scripts into a package in Enterprise Architect. Parameters: <ul style="list-style-type: none"> <li>• PackageGuid: String - the GUID of the package into which to import the build scripts.</li> <li>• BuildScriptXML: String - the build script XML data, which you can export from within Enterprise Architect.</li> </ul>
<b>ImportTechnology (string Technology)</b>	<i>Boolean</i>	Installs a given MDG Technology resource into the repository. Returns <b>True</b> , if the technology is successfully loaded into the model. Otherwise returns <b>False</b> . <b>Note:</b> This applies to technologies imported into pre-7.0 versions of Enterprise Architect (imported technologies), not to technologies referenced in version 7.0 and later (referenced technologies). See <a href="#">Deploying MDG Technologies</a> <sup>[58]</sup> (from Add-Ins). Parameters: <ul style="list-style-type: none"> <li>• Technology: String - the contents of the technology resource file.</li> </ul>
<b>IsTabOpen (string TabName)</b>	<i>String</i>	Checks whether a named Enterprise Architect tabbed view is open and active. This includes open diagram windows or custom controls added using <a href="#">Repository.AddTab()</a> <sup>[193]</sup> . Returns: <ul style="list-style-type: none"> <li>• <b>2</b> to indicate that a tab is open and active (top-most)</li> <li>• <b>1</b> to indicate that it is open but not top-most, or</li> <li>• <b>0</b> to indicate that it is not visible at all.</li> </ul> <b>Note:</b> TabName is case-sensitive.



Method	Type	Notes
		Parameters: <ul style="list-style-type: none"> <li>TabName: String - the name of the tab to check for.</li> </ul>
<b>IsTechnologyEnabled</b> (string ID)	<i>Boolean</i>	Checks whether a specified technology is enabled in Enterprise Architect.  Returns <b>True</b> if the MDG Technology resource is enabled. Otherwise returns <b>False</b> .  Parameters: <ul style="list-style-type: none"> <li>ID: String - the technology ID to check for.</li> </ul>
<b>IsTechnologyLoaded</b> (string ID)	<i>Boolean</i>	Checks whether a specified technology is loaded into the repository.  Returns <b>True</b> if the MDG Technology resource is loaded into the repository. Otherwise returns <b>False</b> .  Parameters: <ul style="list-style-type: none"> <li>ID: String - the technology ID to check for.</li> </ul>
<b>OpenDiagram</b> (long DiagramID)		Provides a method for an automation client or Add-In to open a diagram. The diagram is added to the tabbed list of open diagrams in the main Enterprise Architect view.  Parameters: <ul style="list-style-type: none"> <li>DiagramID: Long - the ID of the diagram to open.</li> </ul>
<b>OpenFile</b> (string Filename)	<i>Boolean</i>	This is the main point for opening an Enterprise Architect project file from an automation client, and working with the contained objects.  If the required project is a DBMS repository, and you have created a shortcut .EAP file containing the database connection string, you can call this shortcut file to access the DBMS repository.  You can also connect to a SQL database by passing in the connection string itself instead of a filename. A valid connection string can be obtained from the <b>Open Project</b> dialog (see <i>Using Enterprise Architect - UML Modeling Tool</i> ) by selecting a recently opened SQL repository.  Parameters: <ul style="list-style-type: none"> <li>Filename: String - the filename of the Enterprise Architect project to open.</li> </ul>
<b>OpenFile2</b> (string FilePath, string Username, string Password)	<i>Boolean</i>	As for <i>OpenFile()</i> except this enables the specification of a password.  Parameters: <ul style="list-style-type: none"> <li>Filepath: String - the file path of the Enterprise Architect project to open.</li> <li>Username: String - the user login ID</li> <li>Password: String - the user password.</li> </ul>
<b>RefreshModelView</b> (long PackageID)		Reloads a package or the entire model, updating the user interface.  Parameters: <ul style="list-style-type: none"> <li>PackageID: Long - the ID of the package to reload: if <b>0</b>, the entire model is reloaded; if a valid package ID, only that package is reloaded.</li> </ul>
<b>RefreshOpenDiagrams</b>		Refreshes the diagram contents for all diagrams open in

Method	Type	Notes
(boolean FullReload)		Enterprise Architect. Parameters: <ul style="list-style-type: none"> <li>FullReload: Boolean - if <b>false</b> the displayed contents of elements and connectors are refreshed in each diagram; if <b>true</b> each of the diagrams is completely reloaded from the repository.</li> </ul>
ReloadDiagram (long DiagramID)		Reloads a specified diagram. This would commonly be used to refresh a visible diagram after code import/export or other batch process where the diagram requires complete refreshing. Parameters: <ul style="list-style-type: none"> <li>DiagramID: Long - the ID of the diagram to be reloaded.</li> </ul>
RemoveOutputTab (string Name)		Removes a specified tab from the <b>Output</b> window. Parameters: <ul style="list-style-type: none"> <li>Name: String - the name of the tab to be removed.</li> </ul>
RunModelSearch (string sQueryName, string sSearchTerm, string sSearchOptions, string sSearchData)		Runs a search, displaying the results in Enterprise Architect's <b>Model Search</b> window. Parameters: <ul style="list-style-type: none"> <li>sQueryName: String - the name of the search to run, for example <i>Simple</i>.</li> <li>sSearchTerm: String - the term to search for.</li> <li>sSearchOptions: String - currently not being used.</li> <li>sSearchData: String - enables you to supply a list of results in the form of XML, which is appended onto the result list in Enterprise Architect. See <a href="#">XML Format</a><sup>[127]</sup>; this parameter is not mandatory so pass in an empty string to run the search as per normal.</li> </ul>
SaveAllDiagrams ()		Saves all open diagrams.
SaveAuditLogs (string FilePath, object StartDateTime, object EndDateTime)	<i>Boolean</i>	Saves the Audit Logs contained within a model to a specified file. If <i>StartDateTime</i> and <i>EndDateTime</i> are not null then only log items that fall into this period are saved. Returns <b>true</b> for success, <b>false</b> for failure.  <b>Note:</b> This might fail if the user logged into the model does not have the correct access permission.  Parameters: <ul style="list-style-type: none"> <li>FilePath: String - the file to save the Audit Logs to.</li> <li>StartDateTime: Variant [DateTime] - the earliest date and time of log entries to save.</li> <li>EndDateTime: Variant [DateTime] - the latest date and time of log entries to save.</li> </ul>
SaveDiagram (long DiagramID)		Saves an open diagram. Assumes the diagram is open in the main user interface Tab list. Parameters: <ul style="list-style-type: none"> <li>DiagramID: Long - the ID of the diagram to save.</li> </ul>

Method	Type	Notes
<b>ShowDynamicHelp (string Topic)</b>		Shows a help topic as a view. Parameters: <ul style="list-style-type: none"> <li>• Topic: String - specifies the help topic.</li> </ul>
<b>ShowInProjectView (object Item)</b>		Selects a specified object in the <b>Project Browser</b> . Accepted object types are <i>Package</i> , <i>Element</i> , <i>Diagram</i> , <i>Attribute</i> , and <i>Method</i> . An exception is thrown if the object is of an invalid type. Parameters: <ul style="list-style-type: none"> <li>• Item: Object - the object to highlight.</li> </ul>
<b>ShowProfileToolbox (string Technology, string Profile, boolean Show)</b>		Shows/hides the contents of a specified technology or profile in the Enterprise Architect UML <b>Toolbox</b> . To show/hide a profile in the <b>Toolbox</b> , specify the profile's ID value in the <i>Profile</i> parameter and set the <i>Technology</i> parameter to a null string. To show/hide a technology in the <b>Toolbox</b> , specify the technology's ID in the <i>Technology</i> parameter and set the <i>Profile</i> parameter to a null string. Parameters: <ul style="list-style-type: none"> <li>• Technology: String - the ID of the technology.</li> <li>• Profile: String - the ID of the profile.</li> <li>• Show: Boolean - if <b>true</b>, show the technology or profile; if <b>false</b>, hide the technology or profile.</li> </ul>
<b>ShowWindow (long Show)</b>		Shows or hides Enterprise Architect. Parameters: <ul style="list-style-type: none"> <li>• Show: Long.</li> </ul>
<b>SQLQuery (string SQL)</b>	<i>String</i>	Enables execution of a SQL <i>select</i> statement against the current repository. Returns an XML formatted string value of the resulting recordset. Parameters: <ul style="list-style-type: none"> <li>• SQL: String - contains the SQL Select statement.</li> </ul>
<b>WriteOutput (string Name, string String, long ID)</b>		Writes text to a specified tab in the <b>Output</b> window, and associates the text with an ID. See also <a href="#">ClearOutput</a> <sup>[194]</sup> , <a href="#">CreateOutputTab</a> <sup>[195]</sup> , <a href="#">EnsureOutput Visible</a> <sup>[195]</sup> . Parameters: <ul style="list-style-type: none"> <li>• Name: String - specifies the tab on which to display the text.</li> <li>• String: String - specifies the text to display.</li> <li>• ID: Long - specifies the ID the text is associated with.</li> </ul>

### 7.2.4.2 Author

#### public Class

An *Author* object represents a named model author. Accessed using the Repository *Authors* collection.

Associated table in .EAP file: *t\_authors*

### Author Attributes

Attribute	Type	Notes
Name	String	Read/Write. Author name.
Notes	String	Read/Write. Notes about the author.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
Roles	String	Read/Write. Roles the author might play in this project.

### Author Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Updates the current Author object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.4.3 Client

##### public Class

A Client represents one or more people or organizations related to the project. Accessed using the Repository *Clients* collection.

Associated table in .EAP file: *t\_clients*

### Client Attributes

Attribute	Type	Notes
EMail	String	Read/Write. EMail address.
Fax	String	Read/Write. Fax number.
Mobile	String	Read/Write. Mobile phone if available.
Name	String	Read/Write. Client name.
Notes	String	Read/Write. Notes about client.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through the <i>Dispatch</i> interface.
Organization	String	Read/Write. Associated organization.
Phone1	String	Read/Write. Main phone number.
Phone2	String	Read/Write. Second phone number.
Roles	String	Read/Write. Roles this client might play in the project.

## Client Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Updates the current Client object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.4.4 Collection

#### public Class

This is the main collection Class used by all elements within the Automation Interface. It contains methods to iterate through the collection, refresh the collection and delete an item from the collection. It is important to realize that when *AddNew* is called, the item is not automatically added to the current collection. The typical steps are:

1. Call *AddNew* to add a new item.
2. Modify the item as required.
3. Call *Update* on the item to save it to the database.
4. Call *Refresh* on the collection to include it in the current set.

*Delete* is much the same; until *Refresh* is called, the collection still contains a reference to the deleted item, which should not be called.

Each can be used to iterate through the collection for languages that support this type of construct.

#### Collection Attributes

Attribute	Type	Notes
<b>Count</b>	<i>Short</i>	Read only. The number of objects referenced by this list.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small> 188 </small>	Read only. Distinguishes objects referenced through a Dispatch interface.

#### Collection Methods

Method	Type	Notes
<b>AddNew (string Name, string Type)</b>	<i>Object</i>	Adds a new item to the current collection.  Note that the interface is the same for all collections; you must provide a <i>Name</i> and <i>Type</i> argument. What these are used for depends on the actual collection member. Also note that you must call <i>Update()</i> on the returned object to complete the <i>AddNew</i> . If <i>Update()</i> is not called the object is left in an indeterminate state.  Parameters: <ul style="list-style-type: none"> <li>• Name: String</li> <li>• Type: String (up to 30 characters long)</li> </ul>
<b>Delete (short index)</b>	<i>Void</i>	Deletes the item at the selected reference.  Parameters: <ul style="list-style-type: none"> <li>• index: Short</li> </ul>

Method	Type	Notes
<b>DeleteAt (short index, boolean Refresh)</b>	<i>Void</i>	Deletes the item at the selected index. The second parameter is currently unused. Parameters: <ul style="list-style-type: none"> <li>index: Short</li> <li>Refresh: Boolean</li> </ul>
<b>GetAt (short index)</b>	<i>Object</i>	Retrieves the array object using a numerical index. If the index is out of bounds, an error occurs. Parameters: <ul style="list-style-type: none"> <li>index: Short</li> </ul>
<b>GetByName (string Name)</b>	<i>Object</i>	Gets an item in the current collection by Name. If the collection does not contain any items, the method returns a null value. If the collection contains items, but it was unable to find an object with the specified name, the method raises an exception. Only supported for the following collections: Models, <a href="#">Packages</a> <sup>[209]</sup> , <a href="#">Elements</a> <sup>[221]</sup> , <a href="#">Diagrams</a> <sup>[255]</sup> , and element <a href="#">TaggedValues</a> <sup>[233]</sup> . Parameters: <ul style="list-style-type: none"> <li>Name: String</li> </ul>
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used, as an exception is thrown when an error occurs.
<b>Refresh ()</b>	<i>Void</i>	Refreshes the collection by re-querying the model and reloading the collection. Should be called after adding a new item or after deleting an item.

### 7.2.4.5 Datatype

#### public Class

A *Datatype* is a named type that can be associated with attribute or method types. It typically is related to either code engineering or database modeling. Datatypes also indicate which language or database system they relate to. Accessed using the Repository *Datatypes* collection.

Associated table in .EAP file: *t\_datatypes*

#### Datatype Attributes

Attribute	Type	Notes
<b>DatatypeID</b>	<i>Long</i>	Read/Write. Instance ID for this datatype within the current model. System maintained.
<b>DefaultLen</b>	<i>Long</i>	Read/Write. Default length (DDL only).
<b>DefaultPrec</b>	<i>Long</i>	Read/Write. Default precision (DDL only).
<b>DefaultScale</b>	<i>Long</i>	Read/Write. Default scale (DDL only).
<b>GenericType</b>	<i>String</i>	Read/Write. The associated generic type for this data type.
<b>HasLength</b>	<i>String</i>	Read/Write. Indicates datatype has a length component.

Attribute	Type	Notes
MaxLen	Long	Read/Write. Maximum length (DDL only).
MaxPrec	Long	Read/Write. Maximum precision (DDL only).
MaxScale	Long	Read/Write. Maximum scale (DDL only).
Name	String	Read/Write. The datatype name (such as <i>integer</i> ). This appears in the related drop-down datatype lists where appropriate.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Product	String	Read/Write. The datatype product, such as Java, C++, Oracle.
Size	Long	Read/Write. The datatype size.
Type	String	Read/Write. The type can be <i>DDL</i> for database datatype or <i>Code</i> for language datatypes.
UserDefined	Long	Read/Write. Indicates if datatype is a user defined type or system generated.  Datatypes distributed with Enterprise Architect are all system generated. Datatypes created in the <b>Datatype</b> dialog are marked <b>1 (true)</b> .

### Datatype Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Updates the current Datatype object after modification or appending a new item.  If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.4.6 EventProperties

An *EventProperties* object is passed to *BroadcastFunctions* to facilitate parameter passing.

### EventProperties Attributes

Attribute	Type	Notes
Count	Long	Read only. Number of parameters being passed to this broadcast event.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.

### EventProperties Methods

Method	Type	Notes
Get (object Index)	<a href="#">EventProperty</a> <small>[208]</small>	Read only. Returns an <i>EventProperty</i> in the list, raising an error if <i>Index</i> is out of range.

Method	Type	Notes
		Parameters: <ul style="list-style-type: none"> <li>Index: Variant - can either be a number representing a zero-based index into the array, or a string representing the name of the <i>EventProperty</i>. For example, <i>Props.Get(3)</i> or <i>Props.Get("ObjectID")</i>.</li> </ul>

#### 7.2.4.7 EventProperty

*EventProperty* objects are always part of an [EventProperties](#)<sup>[207]</sup> collection, and are passed to Add-In methods responding to [broadcast events](#)<sup>[133]</sup>.

#### EventProperty Attributes

Attribute	Type	Notes
Description	<i>String</i>	Explanation of what this property represents.
Name	<i>String</i>	A string distinguishing this property from others in the list.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Distinguishes objects referenced through a Dispatch interface.
Value	<i>Variant</i>	A string, number or object reference representing the property value.

#### 7.2.4.8 ModelWatcher

##### public Class

The *ModelWatcher* object enables an automation client to track changes in a particular model.

##### Note:

After your model has been loaded, you only create the *ModelWatcher* once. If you reload the model, or load another model, the created *ModelWatcher* is still valid.

#### ModelWatcher Attributes

Attribute	Type	Notes
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.

#### ModelWatcher Methods

Methods	Type	Notes
<b>GetReloadItem (object Item)</b>	<a href="#">ReloadType</a> <sup>[189]</sup>	The object that must be reloaded in order to see all changes is returned through the <i>Item</i> parameter. If there are no changes or the entire model must be reloaded, this value is returned as <b>null</b> (C#) or <b>Nothing</b> (VB).  Calling this method clears the records so that the next time it is called the return values refer only to new changes. Returns a value from the <i>ReloadType</i> enumeration that specifies which type of change, if any, has occurred.  Parameters: <ul style="list-style-type: none"> <li>Item: Object</li> </ul>



Methods	Type	Notes
PeekReloadItem	<a href="#">ReloadType</a> <small>[189]</small>	This method behaves identically to <i>GetReloadItem()</i> but does not clear the change record.

### 7.2.4.9 Package

#### public Class

A *Package* object corresponds to a Package element in the Enterprise Architect **Project Browser**. It is accessed either through the Repository *Models* collection (a Model is a special form of Package) or through the Package *Packages* collection. Note that a Package has an Element object as an attribute; this corresponds to an Enterprise Architect Package element in the *t\_object* table and is used to associate additional information (such as scenarios and constraints) with the logical package. To set additional information for a package, reference the Element object directly. Also note that if you add a Package to a diagram, you should add an instance of the element (not the Package itself) to the *DiagramObjects* collection for a diagram.

Associated table in .EAP file: *t\_package*

#### Package Attributes

Attribute	Type	Notes
Alias	String	Read only. Alias.
BatchLoad	Long	Read/Write. Flag to indicate that the package is batch loaded during batch import from controlled packages. Not currently used.
BatchSave	Long	Read/Write. Boolean value to indicate whether the package is included in the batch XMI export list or not.
CodePath	String	Read/Write. The path to where associated source code is found. Not currently used.
Connectors	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of connectors.
Created	Date	Read/Write. Date the package was created.
Diagrams	<a href="#">Collection</a> <sup>[205]</sup>	Read only. A collection of diagrams contained in this package.
Element	<a href="#">Element</a> <sup>[221]</sup>	Read only. The associated element object. Use to get/set common information such as Stereotype, Complexity, Alias, Author, Constraints, Tagged Values and Scenarios.
Elements	<a href="#">Collection</a> <sup>[205]</sup>	Read only. A collection of elements that belong to this package.
Flags	String	Read/Write. Extended information about the package.
IsControlled	Boolean	Read/Write. Indicates if the package has been marked as <i>Controlled</i> .
IsModel	Boolean	Read only. Indicates if the package is a model or a package.
IsNamespace	Boolean	Read/Write. <b>True</b> is 'package is a Namespace root'. Use <b>0</b> and <b>1</b> to set <b>False</b> and <b>True</b> .
IsProtected	Boolean	Read/Write. Indicates if the package has been marked as <i>Protected</i> .

Attribute	Type	Notes
IsVersionControlled	Boolean	Read. Indicates whether or not this package is under version control.
LastLoadDate	Date	Read/Write. The date XML was last loaded for the package.
LastSaveDate	Date	Read/Write. The date XML was last saved from the package.
LogXML	Boolean	Read/Write. Indicates if XMI export information is to be logged.
Modified	Date	Read/Write. Date the package was last modified.
Name	String	Read/Write. The name of the package.
Notes	String	Read/Write. Notes about this package.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Owner	String	Read/Write. The package owner when using controlled packages.
PackageGUID	Variant	Read only. The global Package ID. Valid across models.
PackageID	Long	Read only. The local Package ID number. Valid only in this model file.
Packages	<a href="#">Collection</a> <small>[205]</small>	Read only. A collection of contained packages that can be walked through.
ParentID	Long	Read/Write. The ID of the package that is the parent of this one. <b>0</b> indicates this package is a <i>model</i> (that is, it has no parent).
TreePos	Long	Read/Write. The relative position in the tree compared to other packages (use to sort packages).
UMLVersion	String	Read/Write. The UML version for XMI export purposes.
UseDTD	Boolean	Read/Write. Indicates if a DTD is to be used when exporting XMI.
Version	String	Read/Write. The version of the package.
XMLPath	String	Read/Write. The path to where the XML is saved when using controlled packages.

### Package Methods

Method	Type	Notes
ApplyGroupLock (string aGroupName)	Boolean	Applies a group lock to the package object, for the specified group, on behalf of the current user.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.  Parameter: <ul style="list-style-type: none"> <li>aGroupName: String - The name of the security group for which to apply the lock.</li> </ul>
ApplyUserLock ()	Boolean	Applies a user lock to the package object for the current user.

Method	Type	Notes
		Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.
<b>Clone</b>	<i>LDISPATCH</i>	Inserts a copy of the package into the same parent as the original package.  Returns the newly-created package.
<b>FindObject (string DottedID)</b>	<i>LPDISPATCH</i>	Returns a package, element, attribute or operation matching the parameter <i>DottedID</i> . If the <i>DottedID</i> is not found, an error is returned: <i>Can't find matching object</i> .  Parameter: <ul style="list-style-type: none"> <li>DottedID: String - Is in the form <i>object.object.object</i> where <i>object</i> is replaced by the name of a package, element attribute or operation. Examples include <i>MyNamespace.Class1</i>, <i>CStudent.m_Name</i>, <i>MathClass.Doublelt(int)</i>.</li> </ul>
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>ReleaseUserLock ()</b>	<i>Boolean</i>	Removes an existing User or Group lock from the package object.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.
<b>Update ()</b>	<i>Boolean</i>	Update the current package object after modification or appending a new item.  If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.  Note that a package object also has an <i>element</i> component that must be taken into account. The package object contains information about the package attributes such as hierarchy or contents. The element attribute contains information about, for example, Stereotype, Constraints or Files - all the attributes of a typical element.
<b>VersionControlAdd (string ConfigGuid, string XMLFile, string Comment, bool KeepCheckedOut)</b>	<i>Void</i>	Places the package under version control, using the specified Version Control Configuration and the specified XML filename.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.  It is recommended that the package be saved using <i>Update()</i> before calling <i>VersionControlAdd()</i> , so that any outstanding changes are not lost.  Parameters: <ul style="list-style-type: none"> <li>ConfigGuid: String - Name corresponding to the Unique ID of the version control configuration to use.</li> <li>XMLFile: String - Name of the XML file to use for this package. This filename is relative to the Working Copy folder specified for the Config.</li> <li>Comment: String - Log message that is added to the version controlled file's history (where applicable).</li> <li>KeepCheckedOut: Boolean - Specify <b>True</b> to add to version control and keep package checked-out.</li> </ul>

Method	Type	Notes
<b>VersionControlCheckin (string Comment)</b>	<i>Void</i>	Perform checkin of the version controlled package. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information. Parameters: <ul style="list-style-type: none"> <li>Comment: String - Log message that is added to the version controlled file's history (where applicable).</li> </ul>
<b>VersionControlCheckout (string Comment)</b>	<i>Void</i>	Perform checkout of the version controlled package. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information. Parameters: <ul style="list-style-type: none"> <li>Comment: String - Log message that is added to the version controlled file's history (where applicable).</li> </ul>
<b>VersionControlGetStatus ()</b>	<i>Long</i>	Returns the version control status of the package. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information. Return value maps to the following enumerated type: <i>enum EnumCheckOutStatus</i> { <i>csUncontrolled</i> = 0, <i>csCheckedIn</i> , <i>csCheckedOutToThisUser</i> , <i>csReadOnlyVersion</i> , <i>csCheckedOutToAnotherUser</i> , <i>csOfflineCheckedIn</i> , <i>csCheckedOutOfflineByUser</i> , <i>csCheckedOutOfflineByOther</i> , <i>csDeleted</i> , }; <i>csUncontrolled</i> - Either unable to communicate with the version control provider associated with the package or the package file is unknown to the provider. <i>csReadOnlyVersion</i> - Package is marked as read-only. An earlier revision of the package has been retrieved from version control. <i>csOfflineCheckedOutToThisUser</i> - Indicates that the package was "checked out" by this user whilst disconnected from version control. <i>csOfflineNotCheckedOutToThisUser</i> - Indicates that Enterprise Architect can not currently connect to the version control config and the package was not previously checked out to this user. <i>csDeleted</i> - The package file has been deleted from version control.
<b>VersionControlRemove ()</b>	<i>Void</i>	Removes version control from the package. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.

#### 7.2.4.10 ProjectIssues

##### public Class

A system-level Issue. Indicates a problem or risk associated with the system as a whole. Accessed using the

Repository *Issues* collection.

Associated table in .EAP file: *t\_issues*

### ProjectIssues Attributes

Attribute	Type	Notes
Category	String	Read/Write. The category this issue belongs to.
Date	Date	Read/Write. Date created.
DateResolved	Date	Read/Write. Date issue resolved.
Name	String	Read/Write. Issue name (that is, the issue itself).
IssueID	Long	Read only. The ID of this issue.
Notes	String	Read/Write. Associated description of issue.
ObjectType	<a href="#">ObjectType</a>  188	Read only. Distinguishes objects referenced through a Dispatch interface.
Owner	String	Read/Write. Owner of issue.
Priority	String	Read/Write. Issue priority. Generally should use Low, Medium or High.
Resolution	String	Read/Write. Description of resolution.
Resolver	String	Read/Write. Person resolving issue.
Severity	String	Read/Write. Issue severity. Should be marked as Low, Medium or High.
Status	String	Read/Write. Current issue status.

### ProjectIssues Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current Issue object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.4.11 ProjectResource

##### public Class

A *Project Resource* is a named person who is available to work on the current project in any capacity. Accessed using the Repository *Resources* collection.

Associated table in .EAP file: *t\_resources*

### ProjectResource Attributes

Attribute	Type	Notes
Email	String	Email address.
Fax	String	Fax number.
Mobile	Variant	Mobile number if available.
Name	String	Name of resource.
Notes	String	A description if appropriate.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.
Organization	<a href="#">Package</a> <sup>[209]</sup> : String	Organization resource associated with.
Phone1	Variant	Main phone.
Phone2	Variant	Alternative phone.
Roles	String	The roles this resource can play in the current project.

### ProjectResource Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current Resource object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.4.12 PropertyType

##### public Class

A *PropertyType* object represents a defined property that can be applied to UML elements as a Tagged Value. Accessed using the Repository *PropertyTypes* collection. Each *PropertyType* corresponds to one of the predefined Tagged Values for the model.

Associated table in .EAP file: *t\_propertytypes*

##### Author Attributes

Attribute	Type	Notes
Description	String	Read/Write. Short description for the property.
Detail	String	Read/Write. Configuration information for the property.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.
Tag	String	Read/Write. Name of the property (Tag Name).

## Author Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>PropertyType</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.4.13 Reference

#### public Class

This Interface provides access to the various lookup tables within Enterprise Architect. Use the Repository *GetReferenceList()* method to get a handle to a list. Valid lists are:

- Diagram
- Element
- Constraint
- Requirement
- Connector
- Status
- Cardinality
- Effort
- Metric
- Scenario
- Status
- Test

#### Reference Attributes

Attribute	Type	Notes
<b>Count</b>	<i>Short</i>	Count of items in the list.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Type</b>	<i>String</i>	The list type (for example, Diagram Types).

#### Reference Methods

Method	Type	Notes
<b>GetAt (short Index)</b>	<i>String</i>	Get the item at the specified index.  Parameters: <ul style="list-style-type: none"> <li>• Index: Short - The index of the item to retrieve from the list.</li> </ul>
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.

Method	Type	Notes
Refresh ()	Short	Refresh the current list and return the count of items.

#### 7.2.4.14 Stereotype

##### public Class

The *Stereotype* element corresponds to a UML stereotype, which is an extension mechanism for varying the behavior and type of a model element. Use the Repository *Stereotypes* collection to add new elements and delete existing ones.

Associated table in .EAP file: *t\_stereotypes*

##### Stereotype Attributes

Attribute	Type	Notes
AppliesTo	String	Read/Write. A reference to the stereotype <i>Base Class</i> , that is, which element it applies to.
MetafileLoadPath	String	Read/Write. Path to an associated metafile. The automation interface does not yet support loading metafiles. To do this you must use the <b>Stereotype</b> tab of the <b>UML Types</b> dialog in Enterprise Architect.
Notes	String	Read/Write. Notes about the stereotype.
Name	String	Read/Write. The stereotype name. Appears in the <b>Stereotype</b> drop list for elements that match the <i>AppliesTo</i> attribute.
ObjectType	<a href="#">ObjectType</a>  188	Read only. Distinguishes objects referenced through a Dispatch interface.
StereotypeGUID	String	Read/Write. Unique identifier for stereotype, generally set and maintained by Enterprise Architect.
Style	String	Read/Write. Additional style specifier for stereotype.
VisualType	String	Read/Write. Indicates an inbuilt visual style associated with a stereotype. Not currently implemented.

##### Stereotype Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current stereotype object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.4.15 Task

##### public Class

A Task is an entry in the System ToDo list. Accessed using the Repository *Tasks* collection.



Associated table in .EAP file: *t\_tasks*

### Task Attributes

Attribute	Type	Notes
<b>ActualTime</b>	<i>Long</i>	Read/Write. Time already expended on task, in hours, days or other units.
<b>AssignedTo</b>	<i>String</i>	Read/Write. Person this task is assigned to; that is, the responsible resource.
<b>EndDate</b>	<i>Date</i>	Read/Write. Date task scheduled to finish.
<b>History</b>	<i>String</i>	Read/Write. Memo field to hold, for example, task history or notes.
<b>Name</b>	<i>Variant</i>	Read/Write. Task name.
<b>Notes</b>	<i>Variant</i>	Read/Write. Description of the task.
<b>ObjectType</b>	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Owner</b>	<i>String</i>	Read/Write. The task owner.
<b>Percent</b>	<i>Long</i>	Read/Write. Percent the task is complete.
<b>Phase</b>	<i>String</i>	Read/Write. The phase of the project the task relates to.
<b>Priority</b>	<i>String</i>	Read/Write. Priority associated with this task.
<b>StartDate</b>	<i>Date</i>	Read/Write. Date task is to start.
<b>Status</b>	<i>Variant</i>	Read/Write. Current task status.
<b>TaskID</b>	<i>Long</i>	Read only. Local ID of task.
<b>TotalTime</b>	<i>Long</i>	Read/Write. The total expected time the task might run - in hours, days or some other unit.
<b>Type</b>	<i>String</i>	Read/Write. Sets or returns string representing the type.

### Task Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current Task object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.4.16 Term

##### public Class

A *Term* object represents one entry in the system glossary. Accessed using the Repository *Terms* collection.

Associated table in .EAP file: *t\_glossary*

### Term Attributes

Attribute	Type	Notes
Meaning	String	Read/Write. The description of the term; its meaning.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Term	String	Read/Write. The glossary item name.
TermID	Long	Read only. A local ID number to identify the term in the model.
Type	String	Read/Write. The type this term applies to (for example, business or technical).

### Term Methods

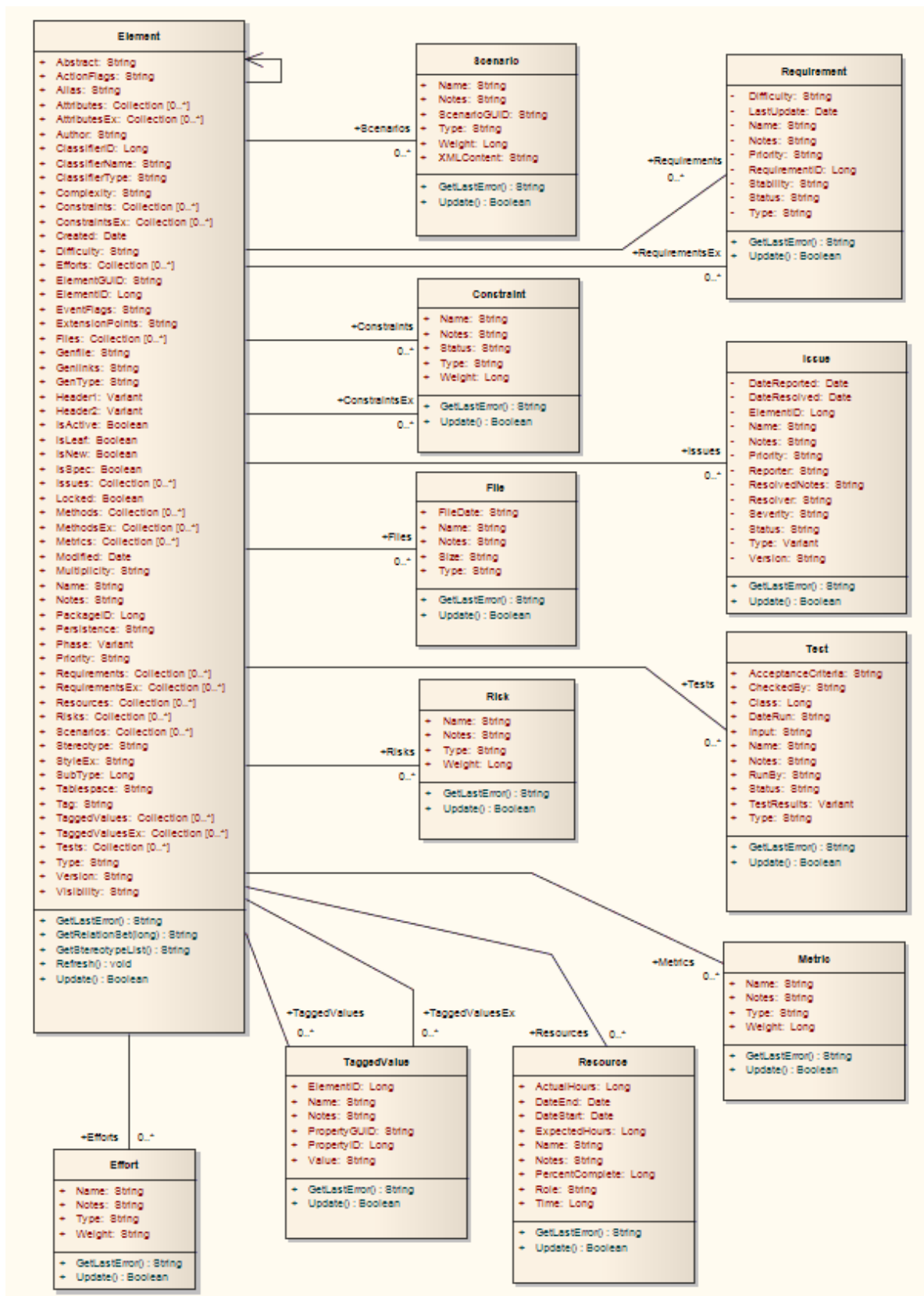
Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current Term object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

## 7.2.5 Element

### public Package

The *Element* package contains information about an element and its associated extended properties such as testing and project management information. An element is the basic item in an Enterprise Architect model. Classes, Use Cases and Components are all different types of UML element.

The diagram below illustrates the relationships between an *element* and its associated extended information. The related information is accessed through the collections owned by the element (for example, Scenarios and Tests). It also includes a full description of the element object (the basic model structural unit).



### 7.2.5.1 Constraint

#### public Class

A *Constraint* is a condition imposed on an element. Constraints are accessed through the Element *Constraints* collection.

Associated table in .EAP file: *t\_objectconstraints*

#### Constraint Attributes

Attribute	Type	Notes
Name	String	Read/Write. The name of the constraint (that is, the constraint).
Notes	String	Read/Write. Notes about the constraint.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
ParentID	Long	Read only. The <i>ElementID</i> of the element to which this constraint applies.
Status	String	Read/Write. Current status.
Type	String	Read/Write. Constraint type.
Weight	Long	Read/Write. A weighting factor.

#### Constraint Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current <i>Constraint</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.2 Effort

#### public Class

An *Effort* is a named item with a weighting that can be associated with an element for purposes of building metrics about the model. Accessed through the Element *Efforts* collection.

Associated table in .EAP file: *t\_objecteffort*

#### Effort Attributes

Attribute	Type	Notes
Name	String	Read/Write. The name of the effort.
Notes	String	Read/Write. Notes about the effort.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.

Attribute	Type	Notes
Type	String	Read/Write. The effort type.
Weight	Long	Read/Write. A weighting factor.
Weight2	Float	Read/Write. A weighting factor.

### Effort Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Saves the effort to the model.

### 7.2.5.3 Element

#### public Class

An *Element* is the main modeling unit. It corresponds to (for example) Class, Use Case, Node or Component. You create new elements by adding to the Package *Elements* collection. Once you have created an element, you can add it to the *DiagramObjects* collection of a diagram to include it in the diagram.

Elements also have a collection of connectors. Each entry in this collection indicates a relationship to another element.

There are also some extended collections for managing additional information about the element, including things such as Tagged Values, Issues, Constraints and Requirements.

Associated table in .EAP file: *t\_object*

#### Element Attributes

Attribute	Type	Notes
Abstract	String	Read/Write. Indicates if the element is Abstract (1) or Concrete (0).
ActionFlags	String	Read/Write. A structure to hold flags concerned with Action semantics.
Alias	String	Read/Write. An optional alias for this element.
Attributes	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Attribute objects for current element. Use the <b>AddNew</b> and <b>Delete</b> functions to manage attributes.
AttributesEx	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Attribute objects belonging to the current element and its parent elements.
Author	String	Read/Write. The element author (see the <a href="#">Repository: Authors</a> <sup>[190]</sup> list for more details).
BaseClasses	<a href="#">Collection</a> <sup>[205]</sup>	Read only. List of Base Classes for this element presented as a collection for convenience.
ClassifierID	Long	<b>Deprecated.</b> See <i>ClassifierID</i> .
ClassifierID	Long	Read/Write. ElementID of a Classifier associated with this

Attribute	Type	Notes
		element; that is, the base type. Only valid for instance type elements (such as Object, Sequence).
<b>ClassifierName</b>	<i>String</i>	Read/Write. Name of associated Classifier (if any).
<b>ClassifierType</b>	<i>String</i>	Read only. Type of associated classifier.
<b>Complexity</b>	<i>String</i>	Read/Write. A complexity value indicating how difficult the element is. Can be used for metric reporting and estimation. Valid values are: <b>1</b> for Easy, <b>2</b> for Medium, <b>3</b> for Difficult.
<b>CompositeDiagram</b>	<a href="#">Diagram</a> <sup>[255]</sup>	Read only. If the element is Composite, returns its associated diagram; otherwise returns null.
<b>Connectors</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Returns a collection containing the connectors to other elements.
<b>Constraints</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of <a href="#">Constraint</a> <sup>[220]</sup> objects.
<b>ConstraintsEx</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Constraint objects belonging to the current element and its parent elements.
<b>Created</b>	<i>Date</i>	Read/Write. The date the element was created.
<b>CustomProperties</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. List of advanced properties for an element. The collection of advanced properties differs depending on element type; for example, an Action and an Activity have different advanced properties. Currently only editable from the user interface.
<b>Diagrams</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Returns a collection of sub-diagrams (child diagrams) attached to this element as seen in the tree view.
<b>Difficulty</b>	<i>String</i>	Read/Write. A difficulty level associated with this element for estimation/metrics; only useable for Requirement, Change and Issue element types, otherwise ignored. Valid values are: <b>Low</b> , <b>Medium</b> , <b>High</b> .
<b>Efforts</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of <a href="#">Effort</a> <sup>[220]</sup> objects.
<b>ElementGUID</b>	<i>String</i>	Read only. A globally unique ID for this element; that is, unique across all model files. If you have to set this value manually, you should only do so when the element is first created, and make sure you format the GUID exactly as Enterprise Architect expects.
<b>ElementID</b>	<i>Long</i>	Read only. The local ID of the Element. Valid for this file only.
<b>Elements</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Returns a collection of child elements (sub-elements) attached to this element as seen in the tree view.
<b>EmbeddedElements</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. List of elements that are embedded into this element, such as Ports, Parts, Pins and Parameter Sets.
<b>EventFlags</b>	<i>String</i>	Read/Write. A structure to hold a variety of flags to do with signals or events.
<b>ExtensionPoints</b>	<i>String</i>	Read/Write. Optional extension points for a Use Case as a comma-separated list.
<b>Files</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of <a href="#">File</a> <sup>[228]</sup> objects.
<b>GenFile</b>	<i>String</i>	Read/Write. The file associated with this element for code

Attribute	Type	Notes
		generation and synchronization purposes. Can include macro expansion tags for local conversion to full path.
<b>Genlinks</b>	<i>String</i>	Read/Write. Links to other Classes discovered at code reversing time; Parents and Implements connectors only.
<b>GenType</b>	<i>String</i>	Read/Write. The code generation type; for example, Java, C++, C#, VBNet, Visual Basic, Delphi.
<b>Header1</b>	<i>Variant</i>	Read/Write. A user defined string for inclusion as header in the source files generated.
<b>Header2</b>	<i>Variant</i>	Read/Write. Same as for <b>Header1</b> , but used in the CPP source file.
<b>IsActive</b>	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is active or not. <b>1</b> = True, <b>0</b> = False.
<b>IsLeaf</b>	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is in leaf node or not. <b>1</b> = True, <b>0</b> = False.
<b>IsNew</b>	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is new or not. <b>1</b> = True, <b>0</b> = False.
<b>IsSpec</b>	<i>Boolean</i>	Read/Write. Boolean value indicating whether the element is a specification or not. <b>1</b> = True, <b>0</b> = False.
<b>Issues</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Issue objects.
<b>Locked</b>	<i>Boolean</i>	Read/Write. Indicates if the element has been locked against further change.
<b>MetaType</b>	<i>String</i>	Read only. The element's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
<b>Methods</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Method objects for current element.
<b>MethodsEx</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Method objects belonging to the current element and its parent elements.
<b>Metrics</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of Metric elements for current element.
<b>MiscData</b>	<i>String</i>	Read only. This low-level property provides information about the contents of the <b>PData</b> fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property. <b>MiscData</b> is zero based, therefore: <ul style="list-style-type: none"> <li>• <b>MiscData(0)</b> corresponds to <b>PData1</b></li> <li>• <b>MiscData(1)</b> to <b>PData2</b></li> </ul> and so on.
<b>Modified</b>	<i>Date</i>	Read/Write. The date the element was last modified.
<b>Multiplicity</b>	<i>String</i>	Read/Write. Multiplicity value for this element.

Attribute	Type	Notes
Name	String	Read/Write. The element name; should be unique within the current package.
Notes	String	Read/Write. Further descriptive text about the element.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
PackageID	Long	Read/Write. A local ID for the package containing this element.
ParentID	Long	Read/Write. If this element is a child of another, used to set or retrieve the <i>ElementID</i> of the other element. If not, returns <b>0</b> .
Partitions	<a href="#">Collection</a> [205]	Read only. List of logical partitions into which an element can be divided. Only valid for elements that support partitions, such as Activities and States.
Persistence	String	Read/Write. The persistence associated with this element. Can be <b>Persistent</b> or <b>Transient</b> .
Phase	String	Read/Write. Phase this element scheduled to be constructed in. Any string value.
Priority	String	Read/Write. The priority of this element as compared to other project elements. Only applies to Requirement, Change and Issue types, otherwise ignored.  Valid values are: <b>Low</b> , <b>Medium</b> and <b>High</b> .
Properties	<a href="#">Properties</a> [245]	Returns a list of specialized properties that apply to the element that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
PropertyType	Long	Read/Write. The ElementID of a Type associated with this element. Only valid for Port and Part elements.
Realizes	<a href="#">Collection</a> [205]	Read only. List of Interfaces realized by this element for convenience.
Requirements	<a href="#">Collection</a> [205]	Read only. Collection of <a href="#">Requirement</a> [230] objects.
RequirementsEx	<a href="#">Collection</a> [205]	Read only. Collection of <a href="#">Requirement</a> [230] objects belonging to the current element and its parent elements.
Resources	<a href="#">Collection</a> [205]	Read only. Collection of <a href="#">Resource</a> [231] objects for current element.
Risks	<a href="#">Collection</a> [205]	Read only. Collection of <a href="#">Risk</a> [232] objects.
RunState	String	Read/Write. The object's runstate list as a string.
Scenarios	<a href="#">Collection</a> [205]	Read only. Collection of <a href="#">Scenario</a> [232] objects for current element.
StateTransitions	<a href="#">Collection</a> [205]	Read only. List of State Transitions that an element can support. Applies in particular to Timing elements.
Status	String	Read/Write. Sets or gets the status, such as <b>Proposed</b> or <b>Approved</b> .
Stereotype	String	Read/Write. The primary element stereotype. This is the first of the list of stereotypes you can access using the <i>StereotypeEx</i>



Attribute	Type	Notes								
		attribute.								
<b>StereotypeEx</b>	String	Read/Write. All the applied stereotypes of the element in a comma-separated list.								
<b>StyleEx</b>	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.								
<b>Subtype</b>	Long	<p>Read/Write. A numeric subtype that qualifies the <a href="#">Type</a> <sup>[225]</sup> of the main element. For example:</p> <ul style="list-style-type: none"> <li>For Event: <b>0</b> = Receiver, <b>1</b> = Sender</li> <li>For Class: <b>1</b> = Parameterised, <b>2</b> = Instantiated, <b>3</b> = Both, <b>0</b> = Neither, <b>17</b> = Association Class</li> </ul> <p><b>Note:</b></p> <p>If <b>17</b>, because an Association Class has been created through the user interface, <b>MiscData(3)</b> will contain the ID of the related Association. As MiscData is read-only, you cannot create an Association Class through the Automation Interface.</p> <ul style="list-style-type: none"> <li>For Note: <b>1</b> = Note linked to connector, <b>2</b> = Constraint linked to connector</li> <li>For StateNode: <b>100</b> = ActivityInitial, <b>101</b> = ActivityFinal</li> <li>For Activity: <b>0</b> = Activity, <b>8</b> = composite Activity (also set to <b>8</b> for other composite elements such as Use Cases)</li> <li>For Synchronization: <b>0</b> = Horizontal, <b>1</b> = Vertical.</li> </ul> <p>Note that there are many more Types than indicated in the above examples.</p>								
<b>Tablespace</b>	String	Read/Write. Associated tablespace for a Table element.								
<b>Tag</b>	String	Read/Write. Corresponds to the <b>Keywords</b> field in the Enterprise Architect user interface. See the <i>General Settings</i> topic in <i>UML Modeling with Enterprise Architect – UML Modeling Tool</i> .								
<b>TaggedValues</b>	<a href="#">Collection</a> <sup>[205]</sup> of type <a href="#">TaggedValue</a> <sup>[233]</sup>	Read only. Returns a collection of <a href="#">TaggedValue</a> <sup>[233]</sup> objects.								
<b>TaggedValuesEx</b>	<a href="#">Collection</a> <sup>[205]</sup> of type <a href="#">TaggedValue</a> <sup>[233]</sup>	Read only. Returns a collection of <a href="#">TaggedValue</a> <sup>[233]</sup> objects belonging to the current element and the elements specialized or realized by the current element.								
<b>Tests</b>	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of <a href="#">Test</a> <sup>[234]</sup> objects for current element.								
<b>TreePos</b>	Long	Read/Write. Sets or gets the tree position.								
<b>Type</b>	String	<p>Read/Write. The element type (such as Class, Component).</p> <p>Note that Type is case sensitive inside Enterprise Architect and should be provided with an initial capital (proper case). Valid types are:</p> <table border="1"> <tbody> <tr> <td><b>Action</b></td> <td><b>InteractionOccurrence</b></td> </tr> <tr> <td><b>Activity</b></td> <td><b>InteractionState</b></td> </tr> <tr> <td><b>ActivityPartition</b></td> <td><b>Interface</b></td> </tr> <tr> <td><b>ActivityRegion</b></td> <td><b>InterruptibleActivityRegion</b></td> </tr> </tbody> </table>	<b>Action</b>	<b>InteractionOccurrence</b>	<b>Activity</b>	<b>InteractionState</b>	<b>ActivityPartition</b>	<b>Interface</b>	<b>ActivityRegion</b>	<b>InterruptibleActivityRegion</b>
<b>Action</b>	<b>InteractionOccurrence</b>									
<b>Activity</b>	<b>InteractionState</b>									
<b>ActivityPartition</b>	<b>Interface</b>									
<b>ActivityRegion</b>	<b>InterruptibleActivityRegion</b>									

Attribute	Type	Notes
		<p>Actor Artifact Association Boundary Change Class Collaboration Component Constraint Decision DeploymentSpecification DiagramFrame EmbeddedElement Entity EntryPoint Event ExceptionHandler ExitPoint ExpansionNode ExpansionRegion GUIElement InteractionFragment</p> <p>Issue Node Note Object Package Parameter Part Port ProvidedInterface Report RequiredInterface Requirement Screen Sequence State StateNode Synchronization Text TimeLine UMLDiagram UseCase</p>
<b>Version</b>	<i>String</i>	Read/Write. The version of the element.
<b>Visibility</b>	<i>String</i>	Read/Write. The Scope of this element within the current package.  Valid values are: <b>Public</b> , <b>Private</b> , <b>Protected</b> or <b>Package</b> .

### Element Methods

Method	Type	Notes
<b>ApplyGroupLock (string aGroupName)</b>	<i>Boolean</i>	Applies a group lock to the element object, for the specified group, on behalf of the current user.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.  Parameter: <ul style="list-style-type: none"> <li>aGroupName: String - the name of the user group for which to set the group lock.</li> </ul>
<b>ApplyUserLock ()</b>	<i>Boolean</i>	Applies a user lock to the element object for the current user.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>GetLinkedDocument ()</b>	<i>String</i>	Returns a string value containing the element's linked document contents, in RTF format.  If the element contains no linked document, an empty string is returned.
<b>GetRelationSet</b>	<i>String</i>	Returns a string containing a comma-separated list of

Method	Type	Notes
<b>(EnumRelationSetType Type)</b>		<p>ElementIDs of directly- <i>and indirectly</i>-related elements based on the given type. See <a href="#">EnumRelationSetType</a><sup>[187]</sup>.</p> <p>Recurses using the same relation type on all elements it finds, retrieving all dependencies and sub-dependencies of the current element; for example, <i>Object1</i> depends on <i>Object2</i>, which depends on <i>Object3</i>. Therefore this method returns <i>Object2 and Object3</i>.</p> <p>To obtain <u>only</u> the direct relationships of the element, use the <a href="#">Connector</a><sup>[246]</sup> collection instead.</p>
<b>GetStereotypeList ()</b>	<i>String</i>	Returns a comma-separated list of stereotypes allied to this element.
<b>LoadLinkedDocument (string Filename)</b>	<i>Boolean</i>	<p>Loads the RTF document from the specified file into the element's linked document.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• FileName: String - the name of the file from which to load the RTF document.</li> </ul>
<b>Refresh ()</b>	<i>Void</i>	Refreshes the element features in the <b>Project Browser</b> . Usually called after adding or deleting attributes or methods, when the user interface is required to be updated as well.
<b>ReleaseUserLock ()</b>	<i>Boolean</i>	<p>Releases a user lock or group lock on the element object.</p> <p>Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.</p>
<b>SaveLinkedDocument (string Filename)</b>	<i>Boolean</i>	<p>Saves the linked document for this element to the specified RTF file.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• FileName: String - the name of the RTF file to which to save the linked document.</li> </ul>
<b>SetAppearance (long Scope, long Item, long Value)</b>	<i>Void</i>	<p>Sets the visual appearance of the element.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>• Scope: Long - Scope of appearance set to modify <ul style="list-style-type: none"> <li><b>0</b> – Local (Diagram-local appearance)</li> <li><b>1</b> – Base (Default appearance across entire model)</li> </ul> </li> <li>• Item: Long - Appearance item to modify <ul style="list-style-type: none"> <li><b>0</b> – Background color</li> <li><b>1</b> – Font Color</li> <li><b>2</b> – Border Color</li> <li><b>3</b> – Border Width</li> </ul> </li> <li>• Value: Long - Value to set appearance to.</li> </ul>
<b>Update ()</b>	<i>Boolean</i>	<p>Update the current element object after modification or appending a new item.</p> <p>If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.</p>

### 7.2.5.4 File

#### public Class

A *File* represents an associated file for an element. It is accessed through the Element *Files* collection.

Associated table in .EAP file: *t\_objectfiles*

#### File Attributes

Attribute	Type	Notes
FileDate	String	Read/Write. The file date when entry is created.
Name	String	Read/Write. The file name can be a logical file or a reference to a web address (using <i>http://</i> ).
Notes	String	Read/Write. Notes about the file.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Size	String	Read/Write. The file size.
Type	String	Read/Write. File type.

#### File Methods

Method	Type	Notes
GetLastError ( )	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ( )	Boolean	Update the current File object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.5 Issue (Maintenance)


#### public Class

An *Issue* is either a *Change* or a *Defect*, is associated with the containing element, and is accessed through the *Issues* collection of an element.

Associated table in .EAP file: *t\_objectproblems*

#### Issue Attributes

Attribute	Type	Notes
DateReported	Date	Read/Write. Date issue reported.
DateResolved	Date	Read/Write. Date issue resolved.
ElementID	Long	Read/Write. ID of element associated with this issue.
Name	String	Read/Write. The Issue name; that is, the Issue itself.
Notes	String	Read/Write. Issue description.
ObjectType	<a href="#">ObjectType</a>	Read only. Distinguishes objects referenced through a Dispatch interface.

Attribute	Type	Notes
		
Priority	String	Read/Write. Issue priority. Generally should use <b>Low</b> , <b>Medium</b> and <b>High</b> .
Reporter	String	Read/Write. Person reporting issue.
Resolver	String	Read/Write. Person resolving issue.
ResolverNotes	String	Read/Write. Notes entered by resolver about resolution.
Severity	String	Read/Write. Issue severity. Should be marked as <b>Low</b> , <b>Medium</b> or <b>High</b> .
Status	String	Read/Write. The current status of the issue.
Type	Variant	Read/Write. Issue type - can be <b>Defect</b> or <b>Change</b> , <b>Issue</b> and <b>ToDo</b> .
Version	String	Read/Write. Version associated with issue. Note that this method is only available through a Dispatch interface. For example: Object ob = Issue; Print ob.Version;

### Issue Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current Issue object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.


### 7.2.5.6 Metric

#### public Class

A *Metric* is a named item with a weighting that can be associated with an element for purposes of building metrics about the model. Accessed through the Element *Metrics* collection.

Associated table in .EAP file: *t\_objectmetrics*

#### Metric Attributes

Attribute	Type	Notes
Name	String	Read/Write. The name of the metric.
Notes	String	Read/Write. Notes about this metric.
ObjectType	<a href="#">ObjectType</a> 	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	String	Read/Write. The metric type.
Weight	Long	Read/Write. A user defined weighting for estimation or metric purposes.

## Metric Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current Metric object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.7 Requirement

#### public Class

An *Element Requirement* object holds information about the responsibilities of an element in the context of the model. Accessed using the *Element Requirements* collection.

Associated table in .EAP file: *t\_objectrequires*

#### Requirement Attributes

Attribute	Type	Notes
<b>Difficulty</b>	<i>String</i>	Read/Write. Estimated difficulty to implement.
<b>LastUpdate</b>	<i>Date</i>	Read/Write. Date requirement last updated.
<b>Name</b>	<i>String</i>	Read/Write. The requirement itself.
<b>Notes</b>	<i>String</i>	Read/Write. Further notes about requirement.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>ParentID</b>	<i>Long</i>	Read only. The <i>ElementID</i> of the element to which this requirement applies.
<b>Priority</b>	<i>String</i>	Read/Write. Assigned priority of the requirement.
<b>RequirementID</b>	<i>Long</i>	Read only. A local ID for this requirement.
<b>Stability</b>	<i>String</i>	Read/Write. Estimated stability of the requirement.  This is an indication of the probability of the requirement - or understanding of the requirement - changing. High stability indicates a low probability of the requirement changing.
<b>Status</b>	<i>String</i>	Read/Write. Current status of the requirement.
<b>Type</b>	<i>String</i>	Read/Write. Requirement type.

#### Requirement Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error

Method	Type	Notes
		occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current Requirement object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.8 Resource

#### public Class

An Element *Resource* is a named person/task pair with timing constraints and percent complete indicators. Use this to manage the work associated with delivering an Element.

Associated table in .EAP file: *t\_objectresources*

#### Resource Attributes

Attribute	Type	Notes
<b>ActualHours</b>	<i>Long</i>	Read/Write. Time already expended on the task, in hours, days or other units.
<b>DateEnd</b>	<i>Date</i>	Read/Write. Expected end date.
<b>DateStart</b>	<i>Date</i>	Read/Write. Date to start work.
<b>ExpectedHours</b>	<i>Long</i>	Read/Write. The total expected time the task might run, in hours, days or other units.
<b>History</b>	<i>String</i>	Read/Write. Gets or sets history text.
<b>Name</b>	<i>String</i>	Read/Write. Name of resource (for example, person's name).
<b>Notes</b>	<i>String</i>	Read/Write. Descriptive notes.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>PercentComplete</b>	<i>Long</i>	Read/Write. Current percent complete figure.
<b>Role</b>	<i>String</i>	Read/Write. Role they play in implementing the element.
<b>Time</b>	<i>Long</i>	Read/Write. Time expected; numeric indicating number of days.

#### Resource Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current Resource object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.9 Risk

#### public Class

A *Risk* object represents a named risk associated with an element and is used for project management purposes. Accessed through the Element *Risks* collection.

Associated table in .EAP file: *t\_objectrisks*

#### Risk Attributes

Attribute	Type	Notes
<b>Name</b>	<i>String</i>	Read/Write. The risk.
<b>Notes</b>	<i>String</i>	Read/Write. Further notes describing the risk.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Type</b>	<i>String</i>	Read/Write. The risk type associated with this element.
<b>Weight</b>	<i>Long</i>	Read/Write. A weighting for estimation or metric purposes.

#### Risk Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current Risk object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.10 Scenario

#### public Class

A *Scenario* corresponds to a Collaboration or Use Case instance. Each scenario is a path of execution through the logic of a Use Case. Scenarios can be added to using the Element *Scenarios* collection.

Associated table in .EAP file: *t\_objectscenarios*

#### Scenario Attributes

Attribute	Type	Notes
<b>Name</b>	<i>String</i>	Read/Write. The Scenario name.
<b>Notes</b>	<i>String</i>	Read/Write. Description of the Scenario. Usually contains the steps to execute the scenario.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>ScenarioGUID</b>	<i>String</i>	Read/Write. A unique ID for the scenario. Used to identify the scenario unambiguously within a model.
<b>Type</b>	<i>String</i>	Read/Write. The Scenario type (for example, <i>Basic Path</i> ).



Attribute	Type	Notes
Weight	Long	Read/Write. Currently used to position scenarios in the scenario list (that is, <i>List Position</i> ).
XMLContent	String	Read/Write. A structured field that can contain scenario details in XML format. <i>Not currently used.</i>

### Scenario Methods

Method	Type	Notes
GetLastError() ( )	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update() ( )	Boolean	Update the current Scenario object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.5.11 TaggedValue

##### public Class

A *TaggedValue* is a named property and value associated with an element and is accessed through the *TaggedValues* collection.

Associated table in .EAP file: *t\_objectproperties*

##### TaggedValue Attributes

Attribute	Type	Notes
ElementID	Long	Read/Write. The local ID of the associated element.
Name	String	Read/Write. Name of the property (Tag).
Notes	String	Read/Write. Further descriptive notes.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String	Read/Write. A global ID for the property.
PropertyID	Long	Read only. A local ID for the property.
Value	String	Read/Write. The value assigned in this instance.

##### TaggedValue Methods

Method	Type	Notes
GetLastError() ( )	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update() ( )	Boolean	Update the current TaggedValue object after modification or appending a

Method	Type	Notes
		new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.5.12 Test

#### public Class

A *Test* is a single Test Case applied to an element. Tests are added and accessed through the Element *Tests* collection.

Associated table in .EAP file: *t\_objecttests*

#### Test Attributes

Attribute	Type	Notes
AcceptanceCriteria	String	Read/Write. The acceptance criteria for successful execution.
CheckedBy	String	Read/Write. Results confirmed by.
Class	Long	Read/Write. The test Class: 1 = Unit Test 2 = Integration Test 3 = System Test 4 = Acceptance Test 5 = Scenario Test.
DateRun	Date	Read/Write. Date last run.
Input	String	Read/Write. Input data.
Name	String	Read/Write. The test name.
Notes	String	Read/Write. Detailed notes about test to be carried out.
ObjectType	<a href="#">ObjectType</a>  1887	Read only. Distinguishes objects referenced through a Dispatch interface.
RunBy	String	Read/Write. Person conducting test.
Status	String	Read/Write. Current status of test.
TestResults	Variant	Read/Write. Results of test.
Type	String	Read/Write. The test type, such as Load or Regression.

#### Test Methods

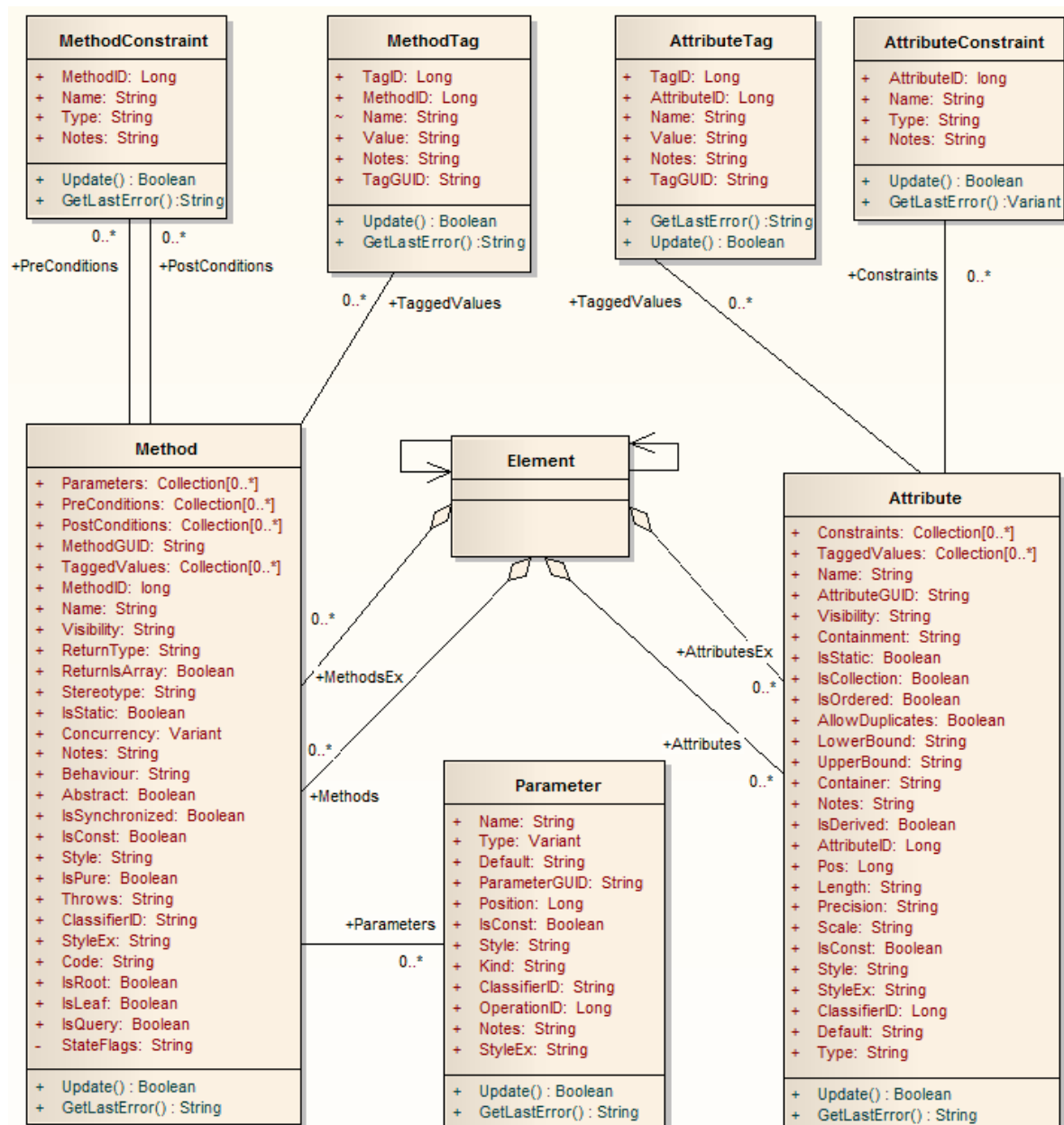
Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current Test object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

## 7.2.6 Element Features

### public Package

The *ElementFeatures* package contains descriptions of the model interfaces that enable access to operations and attributes, and their associated Tagged Values and constraints.

This diagram illustrates the components associated with element features. These include *Attributes* and *Methods*, and the associated constraints and Tagged Values related to them. It also includes the *Parameter* object that defines the arguments associated with an operation (method).



### 7.2.6.1 Attribute

#### public Class

An *attribute* corresponds to a UML Attribute. It contains further collections for constraints and Tagged Values.

Attributes are accessed from the Element *Attributes* collection.

Associated table in .EAP file: *t\_attribute*

### Attribute Attributes

Attribute	Type	Notes
<b>AllowDuplicates</b>	<i>Boolean</i>	Read/Write. Indicates if duplicates are allowed in the collection. If the attribute represents a database column, this when set represents the <b>Not Null</b> option.
<b>AttributeGUID</b>	<i>String</i>	Read/Write. A globally unique ID for the current attribute. System generated.
<b>AttributeID</b>	<i>Long</i>	Read only. Local ID number of the attribute.
<b>ClassifierID</b>	<i>Long</i>	Read/Write. Classifier ID, if appropriate; indicates the base type associated with attribute, if not a primitive type.
<b>Container</b>	<i>String</i>	Read/Write. The container type.
<b>Containment</b>	<i>String</i>	Read/Write. Type of containment. Can be <b>Not Specified, By Reference</b> or <b>By Value</b> .
<b>Constraints</b>	<a href="#">Collection</a> <small>[205]</small>	Read only. A collection of <i>AttributeConstraint</i> objects. Used to access and manage constraints associated with this attribute.
<b>Default</b>	<i>String</i>	Read/Write. Initial value assigned to this attribute.
<b>IsCollection</b>	<i>Boolean</i>	Read/Write. Indicates if the current feature is a collection or not. If the attribute represents a database column, this when set represents a Foreign Key.
<b>IsConst</b>	<i>Boolean</i>	Read/Write. Flag indicating if the attribute is <b>Const</b> or not.
<b>IsDerived</b>	<i>Boolean</i>	Read/Write. Indicates if the attribute is derived (that is, a calculated value).
<b>IsOrdered</b>	<i>Boolean</i>	Read/Write. Indicates if a collection is ordered or not. If the attribute represents a database column, this when set represents a Primary Key.
<b>IsStatic</b>	<i>Boolean</i>	Read/Write. Indicates if the current attribute is a static feature or not. If the attribute represents a database column, this when set represents the <b>Unique</b> option.
<b>Length</b>	<i>String</i>	Read/Write. The attribute length, where applicable.
<b>LowerBound</b>	<i>String</i>	Read/Write. A value for the collection lower bound.
<b>Name</b>	<i>String</i>	Read/Write. The attribute name.
<b>Notes</b>	<i>String</i>	Read/Write. Further notes about this attribute.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>ParentID</b>	<i>Long</i>	Read only. Returns the <i>ElementID</i> of the element that this attribute is a part of.
<b>Pos</b>	<i>Long</i>	Read/Write. Position of the attribute in the Class attribute list.
<b>Precision</b>	<i>String</i>	Read/Write. Precision value.
<b>Scale</b>	<i>String</i>	Read/Write. Scale value.
<b>Stereotype</b>	<i>String</i>	Read/Write. Sets or gets the stereotype for this attribute.

Attribute	Type	Notes
<b>StereotypeEx</b>	<i>String</i>	Read/Write. All the applied stereotypes of the attribute in a comma-separated list.
<b>Style</b>	<i>String</i>	Read/Write. Contains the <b>Alias</b> property for this attribute.
<b>StyleEx</b>	<i>String</i>	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
<b>TaggedValues</b>	<a href="#">Collection</a> [205] of type <a href="#">AttributeTag</a> [238]	Read only. A collection of <i>AttributeTag</i> objects. Use to access and manage Tagged Values associated with this attribute.
<b>TaggedValuesEx</b>	<a href="#">Collection</a> [205] of type <a href="#">TaggedValue</a> [233]	Read only. Collection of <i>TaggedValue</i> objects belonging to the current attribute and the <i>TaggedValuesEx</i> property of its classifier.
<b>Type</b>	<i>String</i>	Read/Write. The attribute type (by name; also see <i>ClassifierID</i> ).
<b>UpperBound</b>	<i>String</i>	Read/Write. A value for the collection upper bound.
<b>Visibility</b>	<i>String</i>	Read/Write. The scope of the attribute. Can be <b>Private</b> , <b>Protected</b> , <b>Public</b> or <b>Package</b> .

### Attribute Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Updates the current attribute object after modifying or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.6.2 AttributeConstraint

#### public Class

An *AttributeConstraint* is a constraint associated with the current Attribute.

Associated table in .EAP file: *t\_attributeconstraints*

#### AttributeConstraint Attributes

Attribute	Type	Notes
<b>AttributeID</b>	<i>Long</i>	Read/Write. ID of the attribute this constraint applies to.
<b>Name</b>	<i>String</i>	Read/Write. The constraint.
<b>Notes</b>	<i>String</i>	Read/Write. Descriptive notes about constraint.
<b>ObjectType</b>	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.

Attribute	Type	Notes
Type	<i>String</i>	Read/Write. Type of constraint.

### AttributeConstraint Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>AttributeConstraint</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.6.3 AttributeTag

#### public Class

An *AttributeTag* represents a Tagged Value associated with an attribute.

Associated table in .EAP file: *t\_attributetag*

#### AttributeTag Attributes

Attribute	Type	Notes
<b>AttributeID</b>	<i>Long</i>	Read/Write. Local ID of attribute associated with this Tagged Value.
<b>Name</b>	<i>String</i>	Read/Write. Name of tag.
<b>Notes</b>	<i>String</i>	Read/Write. Descriptive notes.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>TagGUID</b>	<i>String</i>	Read/Write. A globally unique ID for this Tagged Value.
<b>TagID</b>	<i>Long</i>	Read only. Local ID to identify Tagged Value.
<b>Value</b>	<i>String</i>	Read/Write. Value associated with this tag.

#### AttributeTag Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>AttributeTag</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.6.4 CustomProperties

#### public Collection

The *CustomProperties* collection contains 0 or more *Custom Properties* associated with the current element. These properties provide advanced UML configuration options, and must not be added to or deleted. The value of each property can be set.

#### Note:

The number and type of properties vary depending on the actual element.

#### CustomProperty

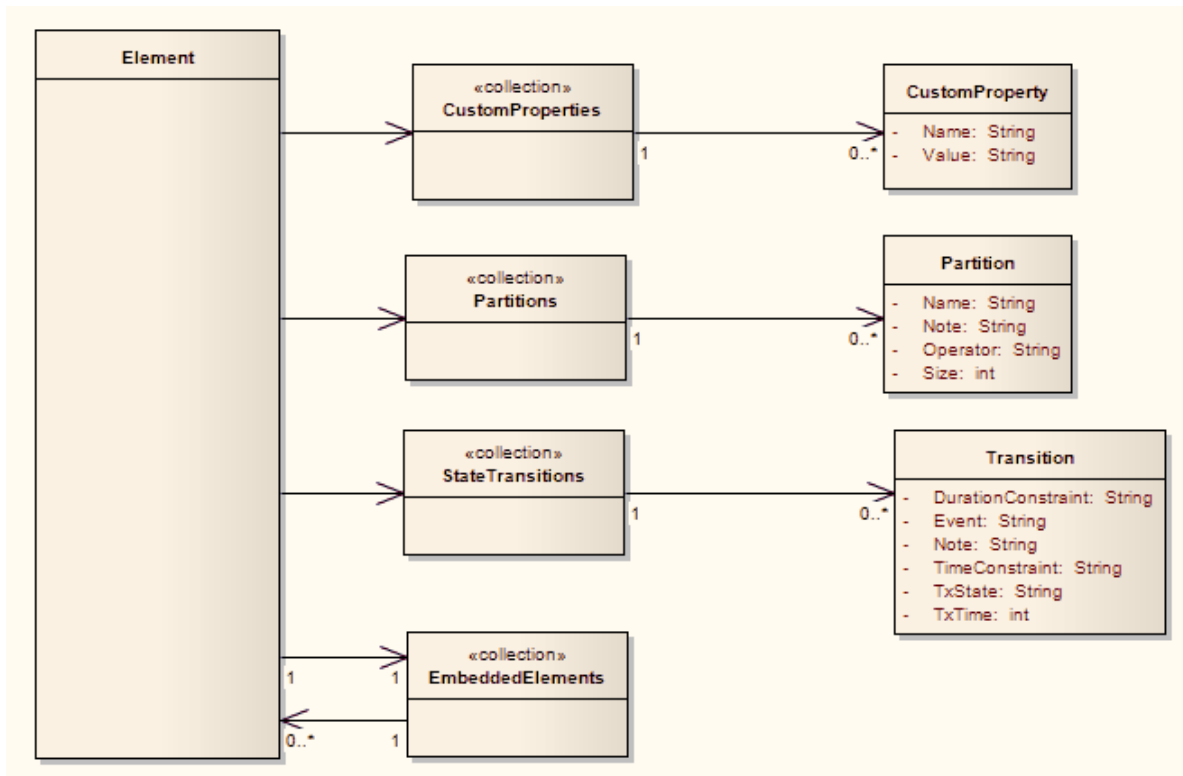
Attribute	Type	Notes
Name	String	Read only. The CustomProperty name.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Value	String	Read/Write. The value associated with this custom property. Can be a string, the boolean values <b>true</b> or <b>false</b> , or an enumeration value from a defined list. The UML 2.1.1 specification in general provides information on enumeration kinds relevant here.

### 7.2.6.5 EmbeddedElements

#### public Collection

In UML 2.1 an element can have one or more embedded elements such as Ports, Pins, Parameters or ObjectNodes. These are attached to the boundary of the host element and cannot be moved off the element. They are owned by their host element. This collection gives easy access to the set of elements embedded on the surface of an element. Note that some embedded elements can have their own embedded element collection (for example, Ports can have Interfaces embedded on them).

The *EmbeddedElements* collection contains Element objects.



### 7.2.6.6 Method

#### public Class

A *method* represents a UML *operation*. It is accessed from the *Element Methods* collection and includes collections for parameters, constraints and Tagged Values.

Associated table in .EAP file: *t\_operation*

#### Method Attributes

Attribute	Type	Notes
<b>Abstract</b>	<i>Boolean</i>	Read/Write. Flag indicating if the method is abstract (1) or not (0).
<b>Behavior</b>	<i>String</i>	Read/Write. Some further explanatory behavior notes (for example, pseudocode).  <b>Note:</b> In earlier releases of Enterprise Architect this attribute had the UK/ Australian spelling 'Behaviour'; this is still present for backwards compatibility, but please now use the 'Behavior' attribute for consistency.
<b>ClassifierID</b>	<i>String</i>	Read/Write. Classifier ID that applies to the <i>ReturnType</i> .
<b>Code</b>	<i>String</i>	Read/Write. Optional field to hold the method Code (used for the <b>Initial Code</b> field).
<b>Concurrency</b>	<i>Variant</i>	Read/Write. Concurrency type of method.
<b>IsConst</b>	<i>Boolean</i>	Read/Write. Flag indicating the method is <b>Const</b> .



Attribute	Type	Notes
IsLeaf	Boolean	Read/Write. Flag to indicate if the method is <i>Leaf</i> (cannot be overridden).
IsPure	Boolean	Read/Write. Flag indicating the method is defined as Pure in C++.
IsQuery	Boolean	Read/Write. Flag to indicate if the method is a query (that is, does not alter Class variables).
IsRoot	Boolean	Read/Write. Flag to indicate if the method is <i>Root</i> .
IsStatic	Boolean	Read/Write. Flag to indicate a static method.
IsSynchronized	Boolean	Read/Write. Flag indicating a Synchronized method call.
MethodGUID	String	Read/Write. A globally unique ID for the current method. System generated.
MethodID	Long	Read only. A local ID for the current method, only valid within this .EAP file.
Name	String	Read/Write. The method name.
Notes	String	Read/Write. Descriptive notes about the method.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
Parameters	<a href="#">Collection</a> [205]	Read only. The <i>Parameters</i> collection for the current method. Use to add and access parameter objects for the current method.
ParentID	Long	Read only. An optional ID of an element that 'owns' this diagram; e.g. a Sequence diagram owned by a Use Case.
Pos	Long	Read/Write. Specifies the position of the method within the set of operations defined for a Class.
PostConditions	<a href="#">Collection</a> [205]	Read only. PostConditions (constraints) as they apply to this method. Returns a <i>MethodConstraint</i> object of type <b>post</b> .
PreConditions	<a href="#">Collection</a> [205]	Read only. PreConditions (constraints) as they apply to this method. Returns a <i>MethodConstraint</i> object of type <b>pre</b> .
ReturnsArray	Boolean	Read/Write. Flag to indicate the return value is an array.
ReturnType	String	Read/Write. Return type for the method; can be a primitive data type or a Class or Interface type.
StateFlags	String	Read/Write. Some flags as applied to methods in State elements.
Stereotype	String	Read/Write. The method stereotype (optional).
StereotypeEx	String	Read/Write. All the applied stereotypes of the method in a comma-separated list.
Style	String	Read/Write. Contains the <b>Alias</b> property for this method.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
TaggedValues	<a href="#">Collection</a> [205] of type <a href="#">MethodTag</a> [243]	Read only. <i>TaggedValues</i> collection for the current method. Accesses a list of <i>MethodTag</i> objects.
Throws	String	Read/Write. Exception information.

Attribute	Type	Notes
Visibility	String	Read/Write. The method scope: <b>Public</b> , <b>Protected</b> , <b>Private</b> or <b>Package</b> .

### Method Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current method object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.6.7 MethodConstraint

##### public Class

A *MethodConstraint* is a condition imposed on a method. It is accessed through either the Method *PreConditions* or Method *PostConditions* collection.

Associated table in .EAP file: *t\_operationpres* and *t\_operationposts*

##### MethodConstraint Attributes

Attribute	Type	Notes
MethodID	Long	Read/Write. The local ID of the associated method.
Name	String	Read/Write. The name of the constraint.
Notes	String	Read/Write. Descriptive notes about this constraint.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	String	Read/Write. The constraint type.

##### MethodConstraint Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current <i>MethodConstraint</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.6.8 MethodTag

#### public Class

A *MethodTag* is a Tagged Value associated with a method.

Associated table in .EAP file: *t\_operationtag*

#### MethodTag Attributes

Attribute	Type	Notes
MethodID	Long	Read/Write. The ID of the associated method.
Name	String	Read/Write. The tag or name of the property.
Notes	String	Read/Write. Descriptive notes about this item.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
TagGUID	String	Read/Write. A unique GUID for this Tagged Value.
TagID	Long	Read only. A unique ID for this Tagged Value.
Value	String	Read/Write. A value to apply to this tag.

#### MethodTag Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current <i>MethodTag</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.6.9 Parameter

#### public Class

A *Parameter* object represents a method argument and is accessed through the *Method Parameters* collection.

Associated table in .EAP file: *t\_operationparams*

#### Parameter Attributes

Attribute	Type	Notes
Alias	String	Read/Write. An optional alias for this parameter.
ClassifierID	String	Read/Write. A ClassifierID for the parameter, if known.
Default	String	Read/Write. A default value for this parameter.
IsConst	Boolean	Read/Write. Flag indicating the parameter is <i>Const</i> (cannot be altered).

Attribute	Type	Notes
Kind	<i>String</i>	Read/Write. The parameter kind - <b>in</b> , <b>inout</b> , <b>out</b> , <b>return</b> .
Name	<i>String</i>	Read/Write. The parameter name; must be unique for a single method.
Notes	<i>String</i>	Read/Write. Descriptive notes.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
OperationID	<i>Long</i>	Read only. ID of the method associated with this parameter.
ParameterGUID	<i>String</i>	Read/Write. A globally unique ID for the current Parameter. System generated.
Position	<i>Long</i>	Read/Write. The position in the argument list.
Stereotype	<i>String</i>	Read/Write. The first stereotype of the parameter.
StereotypeEx	<i>String</i>	Read/Write. All the applied stereotypes of the parameter in a comma-separated list.
Style	<i>String</i>	Read/Write. Some style information.
StyleEx	<i>String</i>	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Type	<i>Variant</i>	Read/Write. The parameter type; can be a primitive type or defined classifier.

### Parameter Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current Parameter object after modifying or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.6.10 Partitions

##### public Collection

A collection of internal element partitions (regions). This is commonly seen in Activity, State, Boundary, Diagram Frame and similar elements (see *The UML Dictionary*). Not all elements support partitions.

This collection contains a set of *Partition* elements. The set is read/write: information is not saved until the host element is saved, so ensure that you call the *Element.Save* method after making changes to a *Partition*.

##### Partition Attributes

Attribute	Type	Notes
Name	<i>String</i>	Read/Write. The partition name; can represent a condition or constraint in some cases.

Attribute	Type	Notes
Note	String	Read/Write. A free text note associated with this partition.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.
Operator	String	Read/Write. An optional operator value that specifies the partition type.
Size	String	Read/Write. Vertical or horizontal width of partition in pixels.

### 7.2.6.11 Properties

#### Properties

##### Properties Attributes

Attribute	Type	Notes
Count	Long	The number of properties that are available for this object.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.

##### Properties Methods

Method	Type	Notes
Item (object Index)	Property	Returns a property either by name or by zero-based integer offset into the list of properties.  Parameter: <ul style="list-style-type: none"> <li>Index: Variant - either a string representing the property name or an integer representing the zero-based offset into the property list.</li> </ul>

#### Property

##### Property Attributes

Attribute	Type	Notes
Name	String	Read only. Identifies the property. The object to which the properties list applies can have an automation property with the same name, in which case the data accessed through <b>Value</b> is identical to that obtained through the automation property.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	<a href="#">PropType</a> <sup>[188]</sup>	Read only. Provides an indication of what sort of data is going to be stored by this property. This restriction can be further defined by the Validation attribute.
Validation	String	Read only. Optional string that is used to validate any data that is passed to the Value attribute. This string is used by the programmer at run time to provide an indication of what's expected, and by Enterprise Architect to ensure that the submitted data is appropriate.
Value	Variant	Read/write. The value of the property as defined in the other fields.

### 7.2.6.12 Transitions

#### public Collection

Applies only to *Timeline elements*. A Timeline element displays 0 or more state transitions at set times on its extent. This collection enables you to access the transition set. You can also access additional information by referring to the connectors associated with the Timeline, and by referencing messages passed between timelines. Note that any changes made to elements in this collection are only saved when the main element is saved.

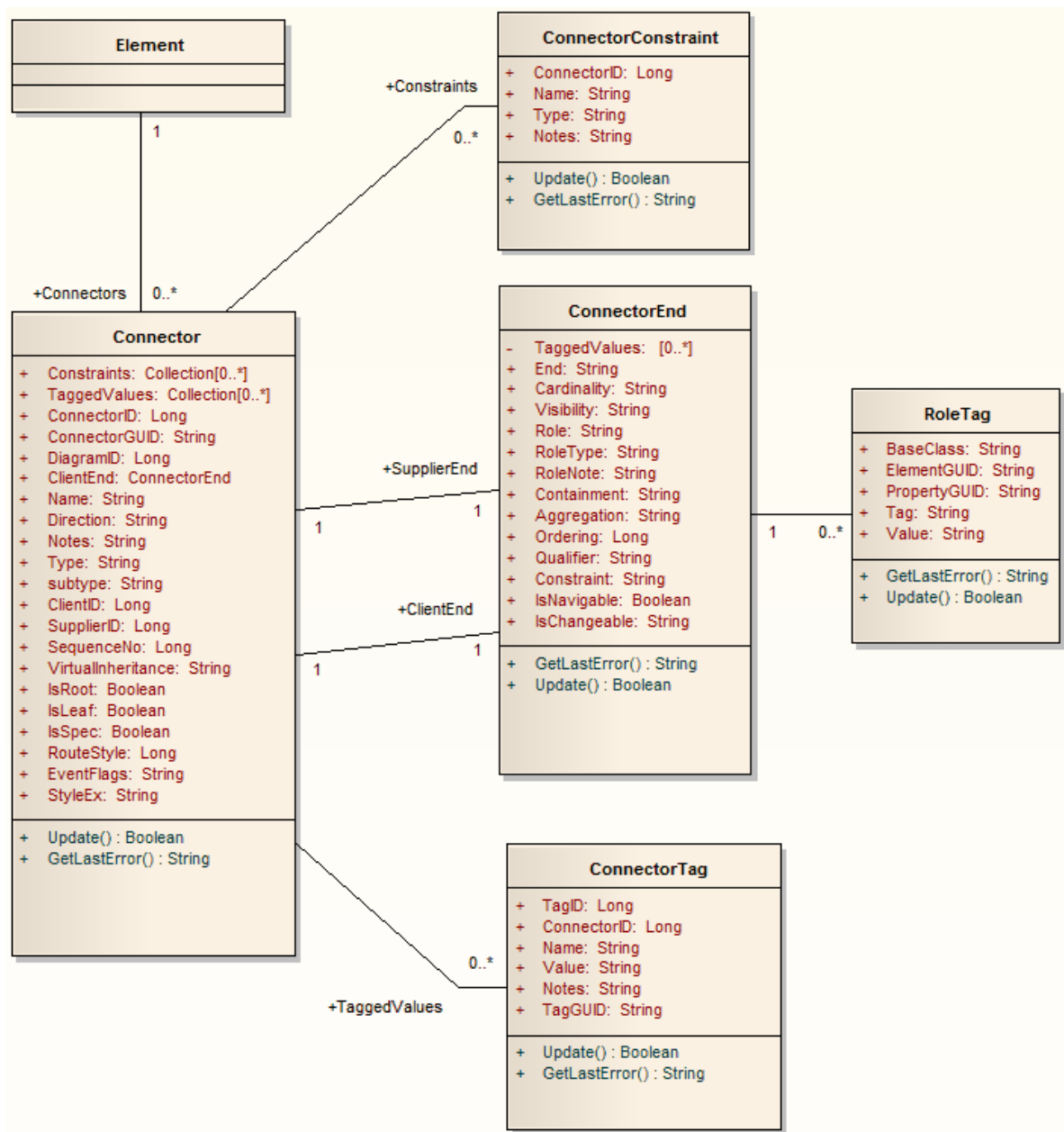
#### Transition Attributes

Attribute	Type	Notes
DurationConstraint	String	Read/Write. A constraint on the time duration that the transition takes.
Event	String	Read/Write. Event (optional) that initiated transition.
Note	String	Read/Write. A free text note.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
TimeConstraint	String	Read/Write. A constraint on when the transition has to be complete by.
TxState	String	Read/Write. The state to transition to. Defined in the <b>Timeline Properties</b> dialog.
TxTime	String	Read/Write. The time that the transition occurs. Value depends on range set in diagram.

### 7.2.7 Connector

#### public Package

The *Connector* package details how connectors between elements are accessed and managed.



### 7.2.7.1 ConnectorConstraint

#### public Class

A *ConnectorConstraint* holds information about special conditions that apply to a connector. It is accessed through the Connector *Constraints* collection.

Associated table in .EAP file: *t\_connectorconstraints*

#### ConnectorConstraint Attributes

Attribute	Type	Notes
ConnectorID	Long	Read/Write. A local ID value (long) - system generated.

Attribute	Type	Notes
Name	String	Read/Write. The constraint name.
Notes	String	Read/Write. Notes about this constraint.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
Type	String	Read/Write. The constraint type.

### ConnectorConstraint Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current <i>ConnectorConstraint</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.7.2 Connector

##### public Class

A *Connector* object represents the various kinds of connectors between UML elements. It is accessed from either the *Client* or *Supplier* element, using the *Connectors* collection of that element. When creating a new connector you must assign it a valid type from the following list:

- Aggregation
- Association
- Collaboration
- Dependency
- Generalization
- Instantiation
- Nesting
- NoteLink
- Realization
- Sequence
- Transition
- UseCase

Associated table in .EAP file: *t\_connector*

##### Connector Attributes

Attribute	Type	Notes
Alias	String	Read/Write. An optional alias for this connector.
ClientEnd	<a href="#">ConnectorEnd</a> [25]	Read only. A pointer to the <i>ConnectorEnd</i> object representing the source end of the relationship.
ClientID	Long	Read/Write. <i>ElementID</i> of the element at the source end of this connector.
Color	Long	Read/Write. Sets the color of the connector.



Attribute	Type	Notes
ConnectorGUID	Variant	Read only. A globally unique ID for the current connector. System generated.
ConnectorID	Long	Read only. Local identifier for the current connector. System generated.
Constraints	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Collection of <a href="#">constraint</a> <sup>[220]</sup> objects.
CustomProperties	<a href="#">Collection</a> <sup>[205]</sup>	Read only. Returns a collection of advanced properties associated with an element in the form of <a href="#">CustomProperty</a> <sup>[239]</sup> objects.
DiagramID	Long	Read/Write. The <i>DiagramID</i> of the connector.
Direction	String	Read/Write. Connector direction. Can be set to one of the following: <ul style="list-style-type: none"> <li>• Unspecified</li> <li>• Bi-Directional</li> <li>• Source -&gt; Destination</li> <li>• Destination -&gt; Source</li> </ul>
EndPointX	Long	Read/Write. The x-coordinate of the connector's end point.  <b>Note:</b> Connector end points are specified in cartesian coordinates with the origin to the top left of the screen.
EndPointY	Long	Read/Write. The y-coordinate of the connector's end point.  <b>Note:</b> Connector end points are specified in cartesian coordinates with the origin to the top left of the screen.
EventFlags	String	Read/Write. Structure to hold a variety of flags concerned with event signaling on messages.
IsLeaf	Boolean	Read/Write. Flag indicating connector is a <i>leaf</i> .
IsRoot	Boolean	Read/Write. Flag indicating connector is a <i>root</i> .
IsSpec	Boolean	Read/Write. Flag indicating connector is a specification.
MetaType	String	Read only. The connector's domain-specific meta type, as defined by an applied stereotype from an MDG Technology.
Name	String	Read/Write. The connector name.
Notes	String	Read/Write. Descriptive notes about the connector.
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.
Properties	<a href="#">Properties</a> <sup>[245]</sup>	Returns a list of specialized properties that apply to the connector that might not be available using the automation model. The properties are purposely undocumented because of their obscure nature and because they are subject to change as progressive enhancements are made to them.
RouteStyle	Long	Read/Write. The route style.
SequenceNo	Long	Read/Write. The <i>SequenceNo</i> of the connector.

Attribute	Type	Notes
StartPointX	Long	Read/Write. The x-coordinate of the connector's start point.  <b>Note:</b> Connector end points are specified in cartesian coordinates with the origin to the top left of the screen.
StartPointY	Long	Read/Write. The y-coordinate of the connector's start point.  <b>Note:</b> Connector end points are specified in cartesian coordinates with the origin to the top left of the screen.
StateFlags	String	Read/Write. Structure to hold a variety of flags concerned with State signaling on messages, the list delimited by semi-colons.
Stereotype	String	Read/Write. Sets or gets the stereotype for this connector end.
StereotypeEx	String	Read/Write. All the applied stereotypes of the connector in a comma-separated list.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Subtype	String	Read/Write. A possible subtype to refine the meaning of the connector.
SupplierEnd	<a href="#">ConnectorEnd</a> <small>[25†]</small>	Read only. A pointer to the <i>ConnectorEnd</i> object representing the target end of the relationship.
SupplierID	Long	Read/Write. <i>ElementID</i> of the element at the target end of this connector.
TaggedValues	Collection	Read only. Collection of <i>ConnectorTag</i> objects.
TransitionAction	String	Read/Write. See the <i>Transition</i> topic in <i>The UML Dictionary</i> for appropriate values.
TransitionEvent	String	Read/Write. See the <i>Transition</i> topic in <i>The UML Dictionary</i> for appropriate values.
TransitionGuard	String	Read/Write. See the <i>Transition</i> topic in <i>The UML Dictionary</i> for appropriate values.
Type	String	Read/Write. Connector type. Valid types are held in the <i>t_connectortypes</i> table in the .EAP file.
VirtualInheritance	String	Read/Write. For <i>Generalization</i> , indicates if inheritance is virtual.
Width	Long	Read/Write. Specifies the width of the connector.

### Connector Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.

Method	Type	Notes
<b>MiscData (long Index)</b>	<i>String</i>	Read only. This low-level property provides information about the contents of the <b>PData</b> fields. These database fields are not documented and developers must gain understanding of these fields through their own endeavors to use this property.  <b>MiscData</b> is zero based, therefore: <ul style="list-style-type: none"> <li>• <b>MiscData(0)</b> corresponds to <b>PData1</b></li> <li>• <b>MiscData(1)</b> to <b>PData2</b></li> </ul> Parameters: <ul style="list-style-type: none"> <li>• Index: long - the zero based index of the <b>PData</b> field to access.</li> </ul>
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>ConnectorObject</i> after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.7.3 ConnectorEnd

#### public Class

A *ConnectorEnd* contains information about a single end of a connector. A *ConnectorEnd* is accessed from the connector as either the *ClientEnd* or *SupplierEnd*.

Associated table in .EAP file: derived from *t\_connector*

#### ConnectorEnd Attributes

Attribute	Type	Notes
<b>Aggregation</b>	<i>Long</i>	Read/Write. Aggregation as it applies to this end. Valid values are: <b>0</b> = None <b>1</b> = Shared <b>2</b> = Composite.
<b>Alias</b>	<i>String</i>	Read/Write. An optional alias for this connector end.
<b>AllowDuplicates</b>	<i>Boolean</i>	Read/Write. For multiplicities greater than 1, indicates that duplicate entries are possible.
<b>Cardinality</b>	<i>String</i>	Read/Write. Cardinality associated with this end.
<b>Constraint</b>	<i>String</i>	Read/Write. A constraint that can be applied to this connector end.
<b>Containment</b>	<i>String</i>	Read/Write. Containment type applied to this connector end.
<b>Derived</b>	<i>Boolean</i>	Read/Write. Indicates that the value of this end is derived.
<b>DerivedUnion</b>	<i>Boolean</i>	Read/Write. Indicates the value of this role derived from the union of all roles that subset this.
<b>End</b>	<i>String</i>	Read only. The end this <i>ConnectorEnd</i> object applies to: <i>Client</i> or <i>Supplier</i> .
<b>IsChangeable</b>	<i>String</i>	Read/Write. Flag indicating whether this end is changeable or not. Values: <b>frozen</b> , <b>addOnly</b> or <b>none</b> .
<b>IsNavigable</b>	<i>Boolean</i>	Read/Write. Flag indicating this end is navigable from the other end.
<b>Navigable</b>	<i>String</i>	Read/Write. Indicates whether this role of an association is navigable from the opposite classifier. Three values are valid: <i>Navigable</i> , <i>Non-</i>

Attribute	Type	Notes
		<i>Navigable and Unspecified.</i>
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Ordering</b>	<i>Long</i>	Read/Write. Ordering for this connector end.
<b>OwnedByClassifier</b>	<i>Boolean</i>	Read/Write. Indicates this association end corresponds to an attribute on the opposite end of the association.
<b>Qualifier</b>	<i>String</i>	Read/Write. A qualifier that can apply to connector end.
<b>Role</b>	<i>String</i>	Read/Write. The connector end role.
<b>RoleNote</b>	<i>String</i>	Read/Write. Notes associated with the role of this connector end.
<b>RoleType</b>	<i>String</i>	Read/Write. The role type applied to this end of the connector.
<b>Stereotype</b>	<i>String</i>	Read/Write. Sets or gets the stereotype for this connector end.
<b>StereotypeEx</b>	<i>String</i>	Read/Write. All the applied stereotypes of the connector end in a comma-separated list.
<b>TaggedValues</b>	<i>Private</i>	Read only. Collection of <a href="#">RoleTag</a> <small>[253]</small> objects.
<b>Visibility</b>	<i>String</i>	Read/Write. Scope associated with this connector end. Valid types are: <i>Public</i> , <i>Private</i> , <i>Protected</i> and <i>Package</i> .

### ConnectorEnd Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>ConnectorEnd</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.7.4 ConnectorTag

##### public Class

A *ConnectorTag* is a Tagged Value for a connector and is accessed through the Connector *TaggedValues* collection.

Associated table in .EAP file: *t\_connectortag*

##### ConnectorTag Attributes

Attribute	Type	Notes
<b>ConnectorID</b>	<i>Long</i>	Read/Write. The local ID of the associated connector.
<b>Name</b>	<i>String</i>	Read/Write. The tag or name.
<b>Notes</b>	<i>String</i>	Read/Write. Descriptive notes associated with this Tagged Value.

Attribute	Type	Notes
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
TagGUID	String	Read/Write. A globally unique ID for this Tagged Value.
TagID	Long	Read only. A local ID to identify the Tagged Value.
Value	String	Read/Write. A value associated with the tag.

### ConnectorTag Methods

Method	Type	Notes
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
Update ()	Boolean	Update the current <i>ConnectorTag</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

#### 7.2.7.5 RoleTag

##### public Class

This interface provides access to the association Role Tagged Values. Each connector end has a *RoleTag* collection that can be accessed to add, delete and access the RoleTags.

In code you create something that resembles the following (where *con* is a Connector Object):

Code fragment for accessing a RoleTag in VB.NET:

```

client = con.ClientEnd
client.Role = "m_client"
client.Update()
tag = client.TaggedValues.AddNew("tag", "value")
tag.Update()
tag = client.TaggedValues.AddNew("tag2", "value2")
tag.Update()
client.TaggedValues.Refresh()
For idx = 0 To client.TaggedValues.Count - 1
    tag = client.TaggedValues.GetAt(idx)
    Console.WriteLine(tag.Tag)
    client.TaggedValues.DeleteAt(idx, False)
Next
tag = Nothing

```

### RoleTag Attributes

Attribute	Type	Notes
BaseClass	String	Read/Write. Indicates the role end; set to <b>ASSOCIATION_SOURCE</b> or <b>ASSOCIATION_TARGET</b> .
ElementGUID	String	Read/Write. GUID of the connector with which this role tag is associated.
ObjectType	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.
PropertyGUID	String	Read/Write. A system generated GUID to identify the Tagged Value.

Attribute	Type	Notes
Tag	<i>String</i>	Read/Write. The actual tag name.
Value	<i>String</i>	Read/Write. The value associated with this tag.

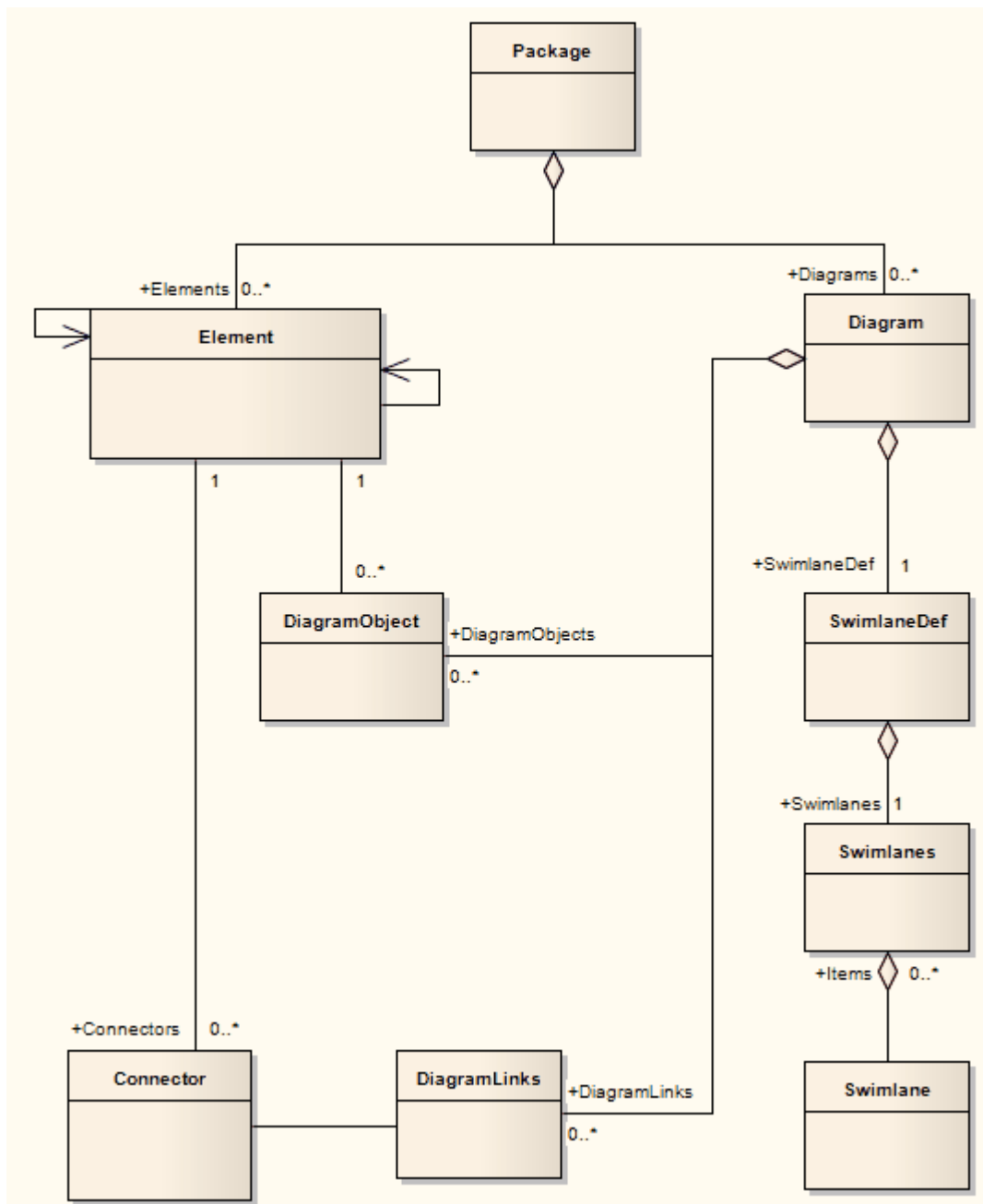
### RoleTag Methods

Method	Type	Notes
<b>GetLastError</b> ( )	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
<b>Update</b> ( )	<i>Boolean</i>	Update the RoleTag after changes or on initial creation. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

## 7.2.8 Diagram

### public Package

The *Diagram* package has information on a diagram and on *DiagramObjects* and *DiagramLinks*, which are the instances of elements within a diagram.



### 7.2.8.1 Diagram

#### public Class

A *Diagram* corresponds to a single Enterprise Architect diagram. It is accessed through the *Package Diagrams* collection and in turn contains a collection of diagram objects and diagram connectors. Adding to the *DiagramObjects* collection adds an element to the diagram (the element must already exist). When adding a new diagram, you must set the diagram type to a valid type; these are:

- Activity
- Analysis
- Component
- Custom
- Deployment

- Logical
- Sequence
- Statechart
- Use Case

**Note:**

Use the Analysis type for a Collaboration Diagram.

Associated table in .EAP file: *t\_diagram*

**Diagram Attributes**

Attribute	Type	Notes
Author	String	Read/Write. The author.
CreatedDate	Date	Read/Write. The date the diagram was created.
cx	Long	Read/Write. The X dimension of the diagram (default is 800).
cy	Long	Read/Write. The Y dimension of the diagram (default is 1100).
DiagramGUID	Variant	Read/Write. A globally unique ID for this diagram.
DiagramID	Long	Read only. A local ID for the diagram.
DiagramLinks	<a href="#">Collection</a> <small>[205]</small>	Read only. A list of <i>DiagramLink</i> objects, each containing information about the display characteristics of a connector in a diagram.  <b>Note:</b> A <i>DiagramLink</i> is only created once a user modifies a connector in a diagram in some way. Until this condition has been met default values are used and the <i>DiagramLink</i> is not in use.
DiagramObjects	<a href="#">Collection</a> <small>[205]</small>	Read only. A collection of references to <a href="#">DiagramObjects</a> <small>[259]</small> . A <i>DiagramObject</i> is an instance of an element in a diagram, and includes size and display characteristics.
ExtendedStyle	String	Read/Write. An extended style attribute.
HighlightImports	Boolean	Read/Write. Flag to indicate elements from other packages should be highlighted.
IsLocked	Boolean	Read/Write. Flag indicating whether this diagram is locked or not.
MetaType	String	Read only. The diagram's domain-specific meta type, as defined by an MDG Technology.
ModifiedDate	Variant	Read/Write. The date the diagram was last modified.
Name	String	Read/Write. The diagram name.
Notes	String	Read/Write. Set/retrieve notes for this diagram.
ObjectType	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
Orientation	String	Read/Write. Page orientation: <b>P</b> for Portrait or <b>L</b> for Landscape.
PackageID	Long	Read/Write. An ID of the package that this diagram belongs to.



Attribute	Type	Notes
ParentID	Long	Read/Write. An optional ID of an element that 'owns' this diagram; for example, a Sequence diagram owned by a Use Case.
Scale	Long	Read/Write. The zoom scale (default is 100).
SelectedConnector	<a href="#">Connector</a> [248]	Read/Write. The currently selected connector on this diagram. Null if there is no currently selected diagram.
SelectedObjects	<a href="#">Collection</a> [205]	Read only. Gets a collection representing the currently selected elements on the diagram. Can remove objects from this collection to deselect them, and add elements to the collection by passing the Object ID as a name to select them.
ShowDetails	Long	Read/Write. Flag to indicate Diagram Details text should be shown. <b>1</b> = Show, <b>0</b> = Hide.
ShowPackageContents	Boolean	Read/Write. Flag to indicate package contents should be shown in the current diagram.
ShowPrivate	Boolean	Read/Write. Flag to show or hide Private features.
ShowProtected	Boolean	Read/Write. Flag to show or hide Protected features.
ShowPublic	Boolean	Read/Write. Flag to show or hide Public features.
Stereotype	String	Read/Write. Sets or gets the stereotype for this diagram.
StyleEx	String	Read/Write. Advanced style settings. Reserved for the use of Sparx Systems.
Swimlanes	String	Read/Write. Information on swimlanes contained in the diagram. Please note that this property is superseded by <a href="#">SwimlaneDef</a> [260].
SwimlaneDef	<a href="#">SwimlaneDef</a> [260]	Read/Write. Information on swimlanes contained in the diagram.
Type	String	Read only. The diagram type. See the <i>t_diagramtypes</i> table in the .EAP file for more information.
Version	String	Read/Write. The version of the diagram.

## Diagram Methods

Method	Type	Notes
ApplyGroupLock (string aGroupName)	Boolean	Applies a group lock to this diagram object, for the specified group, on behalf of the current user.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.  Parameter: <ul style="list-style-type: none"> <li>aGroupName: String - the name of the user group for which to set the group lock.</li> </ul>
ApplyUserLock ()	Boolean	Applies a user lock to this diagram object, for the current user.  Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.
GetLastError ()	String	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error

Method	Type	Notes
		occurs.
<b>ReleaseUserLock ()</b>	<i>Boolean</i>	Releases a group lock or user lock on this diagram object. Throws an exception if the operation fails. Use <i>GetLastError()</i> to retrieve error information.
<b>ReorderMessages ()</b>	<i>Void</i>	Resets the display order of Sequence and Collaboration messages. Typically used after inserting or deleting messages in the diagram.
<b>ShowAsElementList ( bool ShowAsList, bool Persist)</b>	<i>Boolean</i>	Toggles the diagram display between diagram format and <b>Element List</b> depending on the value of <i>ShowAsList</i> . If <i>Persist</i> is set, the display format is written to the database so the diagram always opens in that format (diagram or list). Otherwise, the display format falls back to the default (diagram) once the display is closed. Parameters: <ul style="list-style-type: none"> <li>• ShowAsList: Boolean - indicates diagram or <b>Element List</b></li> <li>• Persist: Boolean - indicates set (maintain <i>ShowAsList</i> value) or not (revert to default).</li> </ul>
<b>Update ()</b>	<i>Boolean</i>	Updates this diagram object after modification or appending a new item. If <b>false</b> is returned, use <i>GetLastError()</i> to retrieve error information.

### 7.2.8.2 DiagramLinks

#### public Class

A *DiagramLink* is an object that holds display information about a connector between two elements in a specific diagram. It includes, for example, the custom points and display appearance. Accessed from the Diagram [DiagramLinks](#)<sup>[258]</sup> collection.

Associated table in .EAP file: *t\_diagramlinks*

#### DiagramLinks Attributes

Attribute	Type	Notes
<b>ConnectorID</b>	<i>Long</i>	Read/Write. The ID of the associated connector.
<b>DiagramID</b>	<i>Long</i>	Read/Write. The local ID for the associated diagram.
<b>Geometry</b>	<i>String</i>	Read/Write. The geometry associated with the current connector in this diagram.
<b>InstanceID</b>	<i>Long</i>	Read only. Holds the connector identifier for the current model.
<b>IsHidden</b>	<i>Boolean</i>	Read/Write. Flag to indicate if this item is hidden or not.
<b>ObjectType</b>	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Path</b>	<i>String</i>	Read/Write. The path of the connector in this diagram.
<b>Style</b>	<i>String</i>	Read/Write. Additional style information; for example, color, thickness.

## DiagramLinks Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used, as an exception is thrown when an error occurs.
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>DiagramLink</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

### 7.2.8.3 DiagramObjects

#### public Class

The *DiagramObjects* collection holds a list of element IDs and presentation information that indicates what is displayed in a diagram and how it is shown.

Associated table in .EAP file: *t\_diagramobjects*

#### DiagramObjects Attributes

Attribute	Type	Notes
<b>Bottom</b>	<i>Long</i>	Read/Write. The bottom position of the element.
<b>DiagramID</b>	<i>Long</i>	Read/Write. The ID of the associated diagram (long).
<b>ElementID</b>	<i>Long</i>	Read/Write. The <i>ElementID</i> of the object instance in this diagram.
<b>InstanceID</b>	<i>Long</i>	Read/Write. Read only attribute. Holds the connector identifier for the current model.
<b>Left</b>	<i>Long</i>	Read/Write. The left position of the element.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Right</b>	<i>Long</i>	Read/Write. The right position of the element.
<b>Sequence</b>	<i>Long</i>	Read/Write. The sequence position when loading into diagram (affects Z order). The Z-order is one-based and the lowest value is in the foreground.
<b>Style</b>	<i>Variant</i>	Write only (reading this value gives undefined results). Style information for this object. See <a href="#">Setting the Style</a> <small>[260]</small> below for more information.
<b>Top</b>	<i>Long</i>	Read/Write. The top position of the element.

#### DiagramObjects Methods

Method	Type	Notes
<b>GetLastError ()</b>	<i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object. This function is rarely used, as an exception is thrown when an error occurs.

Method	Type	Notes
<b>Update ()</b>	<i>Boolean</i>	Update the current <i>DiagramObject</i> object after modification or appending a new item. If <b>false</b> is returned, check the <i>GetLastError</i> function for more information.

## Setting The Style

The *Style* attribute is used for setting the appearance of a *DiagramObject*. It is set with a string value in the format:

```
BCol=n;BFol=n;LCol=n;LWth=n;
```

where:

- *BCol* = Background Color
- *BFol* = Font Color
- *LCol* = Line Color
- *LWth* = Line Width

The color value is a decimal representation of the hex RGB value, where Red=FF, Green=FF00 and Blue=FF0000. For example:

```
DiagObj.Style = "BCol=35723;BFol=9342520;LCol=9342520;LWth=1;"
```

The following code snippet shows how you might change the style settings for all of the objects in the current diagram, in this case changing everything to red.

```
For Each aDiagObj In aDiag.DiagramObjects
    aDiagObj.Style = "BCol=255;BFol=9342520;LCol=9342520;LWth=1;"
    aDiagObj.Update
    aRepos.ReloadDiagram aDiagObj.DiagramID
Next
```

### 7.2.8.4 SwimlaneDef

A *SwimlaneDef* object makes available attributes relating to a single row or column in a list of swimlanes.

Attribute	Type	Notes
<b>Bold</b>	<i>Boolean</i>	Read/Write. Show the title text in bold.
<b>FontColor</b>	<i>Long</i>	Read/Write. RGB color used to draw the titles.
<b>HideClassifier</b>	<i>Boolean</i>	Read/Write. Removes any classifier from title display.
<b>HideNames</b>	<i>Boolean</i>	Read/Write. Set to true to hide the swimlane titles.
<b>LineColor</b>	<i>Long</i>	Read/Write. RGB color used to draw swimlane borders.
<b>LineWidth</b>	<i>Long</i>	Read/Write. Width of line, in pixels, used to draw swimlanes. Valid values: <b>1</b> , <b>2</b> or <b>3</b> .
<b>Locked</b>	<i>Boolean</i>	Read/Write. If set to true, disables user modification of the swimlanes via the diagram.
<b>ObjectType</b>	<a href="#">ObjectType</a>  188	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Orientation</b>	<i>String</i>	Read/Write. Indication of whether the swimlanes are vertical or horizontal.
<b>ShowInTitleBar</b>	<i>Boolean</i>	Read/Write. Enables vertical swimlane titles to be shown in title bar.

Attribute	Type	Notes
Swimlanes	<a href="#">Swimlanes</a> [261]	Read/Write. A list of individual swimlanes.

### 7.2.8.5 Swimlanes

A *Swimlanes* object is attached to a diagram's [SwimlaneDef](#) [260] object and provides a mechanism to access individual swimlanes.

#### Swimlanes Attributes

Attribute	Type	Notes
Count	Long	Read/Write. Gives the number of swimlanes.
<a href="#">ObjectType</a> [188]	<a href="#">ObjectType</a> [188]	Read only. Distinguishes objects referenced through a Dispatch interface.

#### Swimlanes Methods

Method	Type	Notes
<b>Add (string Title, long Width)</b>	<a href="#">Swimlane</a> [262]	Adds a new swimlane to the end of the list. Returns a swimlane object representing the newly added entry.  Parameters: <ul style="list-style-type: none"> <li>Title: String - The title text that appears at the top of the swimlane. Can be the same as an existing swimlane title.</li> <li>Width: Long - The width of the swimlane in pixels.</li> </ul>
<b>Delete (object Index)</b>	Void	Deletes a selected swimlane.  If the string matches more than one entry, only the first entry is deleted.  Parameter: <ul style="list-style-type: none"> <li>Index: Object - Either a string representing the title text or an integer representing the zero-based index of the swimlane to delete.</li> </ul>
<b>DeleteAll ()</b>	Void	Removes all swimlanes.
<b>Insert (long Index, string Title, long Width)</b>	<a href="#">Swimlane</a> [262]	Inserts a swimlane at a specific position. Returns a swimlane object representing the newly added entry.  Parameters: <ul style="list-style-type: none"> <li>Index: Long - The zero-based index of the existing Swimlane before which this new entry is inserted.</li> <li>Title: String - The title text which appears at the top of the swimlane. Can be the same as an existing swimlane title.</li> <li>Width: Long - The width of the swimlane in pixels.</li> </ul>
<b>Items (object Index)</b>	<a href="#">Swimlane</a> [262] <i>collection</i>	Accesses an individual swimlane.  If the string matches more than one swimlane title, the first matching swimlane is returned.  Parameter: <ul style="list-style-type: none"> <li>Index: Object - Either a string representing the title text or an integer representing the zero-based index of the swimlane to get.</li> </ul>

### 7.2.8.6 Swimlane

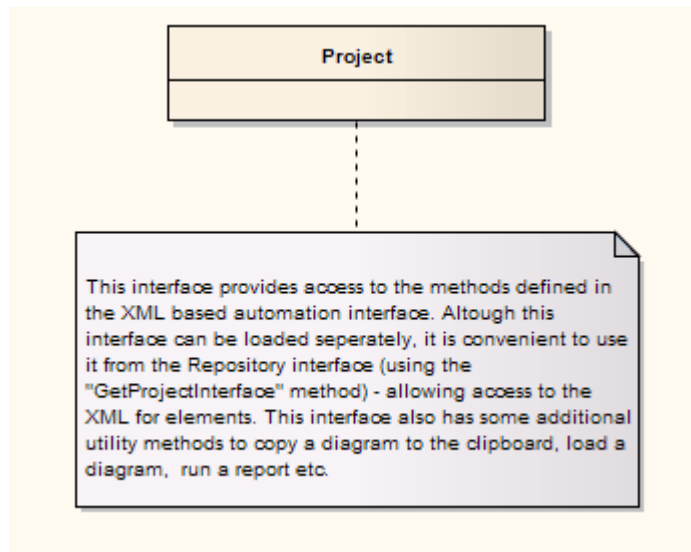
A *Swimlane* object makes available attributes relating to a single row or column in a list of [swimlanes](#)<sup>[267]</sup>.

Attribute	Type	Notes
<b>BackColor</b>	<i>Long</i>	Read/Write. The swimlane is filled with this RGB color.
<b>ClassifiedGuid</b>	<i>String</i>	Read/Write. The GUID of the classifier Class. This can be obtained from the corresponding Element object via the <i>ElementGUID</i> property.
<b>ObjectType</b>	<a href="#">ObjectType</a> <small>[188]</small>	Read only. Distinguishes objects referenced through a Dispatch interface.
<b>Title</b>	<i>String</i>	Read/Write. Text at the head of the swimlane.
<b>Width</b>	<i>Long</i>	Read/Write. The width of the swimlane in pixels.

## 7.2.9 Project Interface

### public Package

The *Enterprise Architect.Project* interface. This is the XML-based interface to Enterprise Architect elements; it also includes some utility functions. You can get a pointer to this interface using the *Repository*. *GetProjectInterface* method.



### 7.2.9.1 Project

#### public Class

The Project interface can be accessed from the Repository using *GetProjectInterface()*. The returned interface provides access to the XML-based Enterprise Architect Automation Interface. Use this interface to get XML for the various internal elements and to run some utility functions to perform tasks such as load diagrams or run reports.

#### Note:

These methods all require input GUIDs in XML format; use [GUIDtoXML](#)<sup>[268]</sup> to change the Enterprise Architect GUID to an XML GUID.

## Project Attributes

Attribute	Type	Notes
ObjectType	<a href="#">ObjectType</a> <sup>[188]</sup>	Read only. Distinguishes objects referenced through a Dispatch interface.

## Project Methods

Method	Type	Notes
CreateBaseline (string PackageGUID, string Version, string Notes)	Boolean	Creates a Baseline of a specified package. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to Baseline.</li> <li>Version: String - the version of the Baseline.</li> <li>Notes: String - any notes concerning the Baseline.</li> </ul>
DefineRule (string CategoryID, EA.EnumMVErrortype ErrorType, string ErrorMessage)	String	Defines the individual rules that can be performed during model validation. It must be called once for each rule from the <a href="#">EA_OnInitializeUserRules</a> <sup>[155]</sup> broadcast handler.  The return value is a <i>RuleId</i> , which can be used for reference purposes when an individual rule is executed by Enterprise Architect during model validation.  See <a href="#">Model Validation Example</a> <sup>[160]</sup> for a detailed example of use of this method.  Parameters: <ul style="list-style-type: none"> <li>CategoryId: String - should be passed the return value from the <a href="#">DefineRuleCategory</a><sup>[263]</sup> method.</li> <li>ErrorType: EA.EnumMVErrortype - depending on the severity of the error being validated, can be: <ul style="list-style-type: none"> <li><b>mvErrorCritical</b></li> <li><b>mvError</b></li> <li><b>mvWarning</b>, or</li> <li><b>mvInformation</b>.</li> </ul> </li> <li>ErrorMessage: String - can contain a default error string, although this is probably overridden by the <a href="#">PublishResult</a><sup>[270]</sup> call.</li> </ul>
DefineRuleCategory (string CategoryName)	String	Defines a category of rules that can be performed during model validation (there is typically one category per Add-In). It must be called once from the <a href="#">EA_OnInitializeUserRules</a> <sup>[155]</sup> broadcast handler.  The return value is a CategoryId that must to be passed to the <a href="#">DefineRule</a> <sup>[263]</sup> method.  See <a href="#">Model Validation Example</a> <sup>[160]</sup> for a detailed example of use of this method.  Parameters: <ul style="list-style-type: none"> <li>CategoryName: String - a text string that is visible in the <b>Model Validation Configuration</b> dialog.</li> </ul>
DeleteBaseline (string BaselineGUID)	Boolean	Deletes a Baseline, identified by the BaselineGUID, from the repository.

Method	Type	Notes
		<p>Parameters:</p> <ul style="list-style-type: none"> <li>BaselineGUID: String - the GUID (in XML format) of the Baseline to delete.</li> </ul>
<b>DoBaselineCompare</b> (string PackageGUID, string Baseline, string ConnectString)	String	<p>Performs a Baseline comparison using the supplied package GUID and Baseline GUID (obtained in the result list from <a href="#">GetBaselines</a><sup>[266]</sup>).</p> <p>Optionally you can include the connection string required to find the Baseline if it exists in a different model file.</p> <p>This method returns a log file of the status of all elements found and compared in the difference procedure. You can use this log information as input to <a href="#">DoBaselineMerge</a><sup>[264]</sup> - automatically merging information from the Baseline.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to run the comparison on.</li> <li>Baseline: String - the GUID (in XML format) of the Baseline to run the comparison on.</li> <li>ConnectString: String - the location of the external .EAP file or DBMS to extract the Baseline from.</li> </ul>
<b>DoBaselineMerge</b> (string PackageGUID, string Baseline, string MergeInstructions, string ConnectString)	String	<p>Performs a batch merge based on instructions contained in an XML file (<i>MergeInstructions</i>). You can supply an optional connection string if the Baseline is located in another model.</p> <p>In the <i>MergeInstructions</i> file, each <i>MergeItem</i> node supplies the GUID of a differenced item from the XML difference log. As the merge is uni-directional and actioned in only one possible way, no additional arguments are required. Enterprise Architect chooses the correct procedure based on the Difference results.</p> <pre>&lt;Merge&gt;   &lt;MergeItem guid="{XXXXXX}" /&gt;   &lt;MergeItem guid="{XXXXXX}" /&gt; &lt;/Merge&gt;</pre> <p>Alternatively, you can supply a single <i>MergeItem</i> with a GUID of <i>RestoreAll</i>. In this case, Enterprise Architect batch-processes ALL differences.</p> <pre>&lt;Merge&gt;   &lt;MergeItem guid="RestoreAll" /&gt; &lt;/Merge&gt;</pre> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to merge the Baseline into.</li> <li>Baseline: String - the GUID of the Baseline (in XML format) to merge into the package.</li> <li>MergeInstructions: String - the file containing the GUID of each differenced item from the XML difference log returned by <a href="#">DoBaselineCompare()</a><sup>[264]</sup>.</li> <li>ConnectString: String - the location of the EAP</li> </ul>



Method	Type	Notes
		file or DBMS to get the Baseline from, if not in the same model as the package.
<b>EnumDiagramElements (string DiagramGUID)</b>	protected abstract: <i>String</i>	Gets an XML list of all elements in a diagram. Parameters: <ul style="list-style-type: none"> <li>DiagramGUID: String - the GUID (in XML format) of the diagram to get elements for.</li> </ul>
<b>EnumDiagrams (string PackageGUID)</b>	protected abstract: <i>String</i>	Gets an XML list of all diagrams in a specified package. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to list diagrams for.</li> </ul>
<b>EnumElements (string PackageGUID)</b>	protected abstract: <i>String</i>	Gets an XML list of elements in a specified package. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to get a list of elements for.</li> </ul>
<b>EnumLinks (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets an XML list of connectors for a specified element. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element to get all associated connectors for.</li> </ul>
<b>EnumPackages (string PackageGUID)</b>	protected abstract: <i>String</i>	Gets an XML list of child packages inside a parent package. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the parent package.</li> </ul>
<b>EnumProjects ()</b>	protected abstract: <i>String</i>	Gets a list of projects in the current file; corresponds to <a href="#">Models</a> <sup>[192]</sup> in <i>Repository</i> .
<b>EnumViews ()</b>	protected abstract: <i>String</i>	Enumerates the Views for a project. Returned as an XML document.
<b>EnumViewEx (string ProjectGUID)</b>	protected abstract: <i>String</i>	Gets a list of Views in the current project. Parameters: <ul style="list-style-type: none"> <li>ProjectGUID: String - the GUID (in XML format) of the project to get views for.</li> </ul>
<b>Exit ()</b>	protected abstract: <i>String</i>	Exits the current instance of Enterprise Architect; this function is maintained for backward compatibility and should never be called.  Enterprise Architect automatically exits when you are no longer using any of the provided objects.
<b>ExportPackageXML (string PackageGUID, enumXMIMType XMIMType, long DiagramXML, long DiagramImage, long FormatXML, long UseDTD, string FileName)</b>	protected abstract: <i>String</i>	Exports XML for a specified package. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to be exported.</li> <li>XMIMType: EnumXMIMType - specifies the XML type and version information; see <a href="#">XMIMType Enum</a><sup>[189]</sup> for accepted values.</li> <li>DiagramXML: Long - <b>true</b> if XML for diagrams is required; accepted values:</li> </ul>

Method	Type	Notes
		<p>0 = Do not export diagrams  1 = Export diagrams  2 = Export diagrams along with alternate images.</p> <ul style="list-style-type: none"> <li>DiagramImage: Long - the format for diagram images to be created at the same time; accepted values:  -1=NONE  0=EMF  1=BMP  2=GIF  3=PNG  4=JPG.</li> <li>FormatXML: Long - <b>true</b> if XML output should be formatted prior to saving.</li> <li>UseDTD: Long - <b>true</b> if a DTD should be used.</li> <li>FileName: String - the filename to output to.</li> </ul>
<b>GenerateClass</b> (string ElementGUID, string ExtraOptions)	<i>Boolean</i>	<p>Generates the code for a single Class.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element to generate.</li> <li>ExtraOptions: String - enables extra options to be given to the command; currently unused.</li> </ul>
<b>GeneratePackage</b> (string Package GUID, string ExtraOptions)	<i>Boolean</i>	<p>Generates the code for all Classes within a package.</p> <p>For example: recurse=1;overwrite=1;dir=C:\</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to generate.</li> <li>ExtraOptions: String - enables extra options to be given to the command; currently enables: <ul style="list-style-type: none"> <li>Generation of all subpackages (<i>recurse</i>)</li> <li>Force overwrite of all files (<i>overwrite</i>) and</li> <li>Specification to auto generate all paths (<i>dir</i>).</li> </ul> </li> </ul>
<b>GenerateXSD</b> (string PackageGUID, string FileName, string Encoding, string Options)	<i>Boolean</i>	<p>Creates an XML schema for this GenerateXSD. Returns <b>true</b> on success.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package.</li> <li>FileName: String - target filepath.</li> <li>Encoding: String - the XML encoding for the code page instruction.</li> <li>Options: String - enables extra options to be given to the command; currently enables: <ul style="list-style-type: none"> <li><i>GenGlobalElement</i> - turn the generation of global elements for all global <i>ComplexTypes</i> <b>On</b> or <b>Off</b>; for example: - GenGlobalElement=1.</li> </ul> </li> </ul>
<b>GetBaselines</b> (string PackageGUID, string ConnectString)	<i>String</i>	<p>Returns a list (in XML format) of Baselines associated with the supplied package GUID. Optionally, you can provide a connection string to get Baselines from the same package, but located in a different model file (or DBMS).</p>

Method	Type	Notes
		Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to get Baselines for.</li> <li>ConnectionString: String - the location of the EAP file or DBMS to get the Baselines from, if not in the same model as the package.</li> </ul>
<b>GetDiagram (string DiagramGUID)</b>	protected abstract: <i>String</i>	Gets diagram details, in XML format. Parameters: <ul style="list-style-type: none"> <li>DiagramGUID: String - the GUID (in XML format) of the diagram to get details for.</li> </ul>
<b>GetElement (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets XML for the specified element. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element to retrieve XML for.</li> </ul>
<b>GetElementConstraints (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets constraints for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementEffort (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets efforts for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementFiles (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets metrics for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementMetrics (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets files for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementProblems (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets a list of issues (problems) associated with an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementProperties (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets Tagged values for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementRequirements (string ElementGUID)</b>	protected abstract: <i>String</i>	Gets a list of requirements for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
<b>GetElementResources (string ElementGUID)</b>	protected	Gets a list of resources for an element, in XML format.

Method	Type	Notes
ElementGUID)	abstract: <i>String</i>	Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
GetElementRisks (string ElementGUID)	protected abstract: <i>String</i>	Gets a list of risks associated with an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
GetElementScenarios (string ElementGUID)	protected abstract: <i>String</i>	Gets a list of scenarios for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
GetElementTests (string ElementGUID)	protected abstract: <i>String</i>	Gets a list of tests for an element, in XML format. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element.</li> </ul>
GetLastError ()	protected abstract: <i>String</i>	Returns a string value describing the most recent error that occurred in relation to this object.  This function is rarely used, as an exception is thrown when an error occurs.
GetLink (string LinkGUID)	protected abstract: <i>String</i>	Gets connector details, in XML format. Parameters: <ul style="list-style-type: none"> <li>LinkGUID: String - the GUID (in XML format) of the connector to get details of.</li> </ul>
GUIDtoXML (string GUID)	<i>String</i>	Changes an internal GUID to the form used in XML. Parameters: <ul style="list-style-type: none"> <li>GUID: String - the Enterprise Architect style GUID to convert to XML format.</li> </ul>
ImportDirectory (string PackageGUID, string Language, string DirectoryPath, string ExtraOptions)	<i>Boolean</i>	Imports a source code directory into the model. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to reverse engineer code into.</li> <li>Language: String - specifies the language of the code to be imported.</li> <li>DirectoryPath: String - specifies the path where the code is found on the computer.</li> <li>ExtraOptions: String - enables extra options to be given to the command; currently enables import of source from all child directories (<i>recurse</i>) - for example: <i>recurse=1</i>.</li> </ul>
ImportFile (string PackageGUID, string Language, string FileName, string ExtraOptions)	<i>Boolean</i>	Imports an individual file or binary module into the model, in a package per namespace style import. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to reverse engineer code into; this is expected to be a namespace root package.</li> <li>Language: String - specifies the language of the</li> </ul>

Method	Type	Notes
		<p>code to be imported.</p> <p><b>Note:</b></p> <p>Use the value "DNPE" to import a binary module. This imports a .Net assembly or Java .class file, but <b>not</b> a .jar file.</p> <ul style="list-style-type: none"> <li>Filename: String - specifies the path where the code or module is found on the computer.</li> <li>ExtraOptions: String - enables extra options to be given to the command; <b>currently unused</b>.</li> </ul>
<b>ImportPackageXML</b> (string PackageGUID, string Filename, long ImportDiagrams, long StripGUID)	<i>String</i>	<p>Imports an XML file at a point in the tree.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the target package to import the XML file into (or overwrite with the XML file).</li> <li>Filename or XMLText: String - the name of the XML file.</li> </ul> <p><b>Note:</b></p> <p>If the String is of type <i>filename</i> it is interpreted as a source file, otherwise the String is imported as XML text.</p> <ul style="list-style-type: none"> <li>ImportDiagrams: Long.</li> <li>StripGUID: Long - boolean value to indicate whether to replace the element UniqueIDs on import; if stripped, then a package could be imported twice into Enterprise Architect, as two different versions.</li> </ul>
<b>LayoutDiagram</b> (string DiagramGUID, long LayoutStyle)	<i>Boolean</i>	<p><b>Deprecated.</b> it is recommended that <i>LayoutDiagramEx</i> is used instead.</p> <p>Calls the function to automatically layout a diagram in hierarchical fashion. It is only recommended for Class and Object diagrams.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>DiagramGUID: String - the GUID (in XML format) of the diagram to lay out.</li> <li>LayoutStyle: Long - always ignored.</li> </ul>
<b>LayoutDiagramEx</b> (string DiagramGUID, long LayoutStyle, long Iterations, long LayerSpacing, long ColumnSpacing, boolean SaveToDiagram)	<i>Boolean</i>	<p>Calls the function to automatically layout a diagram in hierarchical fashion. It is only recommended for Class and Object diagrams.</p> <p><i>LayoutStyle</i> accepts the following options (also see <a href="#">ConstLayoutStyles Enum</a><sup>[186]</sup>):</p> <ul style="list-style-type: none"> <li>Default Options: <ul style="list-style-type: none"> <li>IsDiagramDefault</li> <li>IsProgramDefault.</li> </ul> </li> <li>Cycle Removal Options: <ul style="list-style-type: none"> <li>IsCycleRemoveGreedy</li> <li>IsCycleRemoveDFS.</li> </ul> </li> <li>Layering Options: <ul style="list-style-type: none"> <li>IsLayeringLongestPathSink</li> <li>IsLayeringLongestPathSource</li> </ul> </li> </ul>

Method	Type	Notes
		<p>IsLayeringOptimalLinkLength.</p> <ul style="list-style-type: none"> <li>Initialize Options: <ul style="list-style-type: none"> <li>IsInitializeNaive</li> <li>IsInitializeDFSOut</li> <li>IsInitializeDFSIn.</li> </ul> </li> <li>Crossing Reduction Option: <ul style="list-style-type: none"> <li>IsCrossReduceAggressive.</li> </ul> </li> <li>Layout Options - Direction <ul style="list-style-type: none"> <li>IsLayoutDirectionUp</li> <li>IsLayoutDirectionDown</li> <li>IsLayoutDirectionLeft</li> <li>IsLayoutDirectionRight.</li> </ul> </li> </ul> <p>Parameters:</p> <ul style="list-style-type: none"> <li>DiagramGUID: String - the GUID (in XML format) of the diagram to lay out.</li> <li>LayoutStyle: Long - the layout style.</li> <li>Iterations: Long - the number of layout iterations the Layout process should take to perform cross reduction (Default value = 4).</li> <li>LayerSpacing: Long - the per-element layer spacing the Layout process shall use (Default value = 20).</li> <li>ColumnSpacing: Long - the per-element column spacing the Layout process shall use (Default value = 20).</li> <li>SaveToDiagram: Boolean - specifies whether or not Enterprise Architect should save the supplied layout options as default to the diagram in question.</li> </ul>
<b>LoadControlledPackage (string PackageGUID)</b>	<i>String</i>	<p>Loads a package that has been marked and configured as controlled. The filename details are stored in the package control data.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to load.</li> </ul>
<b>LoadDiagram (string DiagramGUID)</b>	protected abstract: <i>Boolean</i>	<p>Loads a diagram by its GUID.</p> <p>Parameter:</p> <ul style="list-style-type: none"> <li>DiagramGUID: String - the GUID (in XML format) of the diagram to load; if you retrieve the GUID using the Diagram interface, use the <a href="#">GUIDtoXML</a><sup>[268]</sup> function to convert it to XML format.</li> </ul>
<b>LoadProject (string FileName)</b>	protected abstract: <i>Boolean</i>	<p>Loads an Enterprise Architect project file. Do not use this method if you have accessed the Project interface from the Repository, which has already loaded a file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>FileName: String - the name of the project file to load.</li> </ul>
<b>PublishResult (string CategoryID, EA.EnumMVErrortype ErrorType, string ErrorMessage)</b>	<i>String</i>	<p>Returns the results of each rule that can be performed during model validation. It must be called once for each rule from the <a href="#">EA_OnInitializeUserRules</a><sup>[155]</sup> broadcast handler.</p>

Method	Type	Notes
		<p>The return value is a <i>RuleId</i>, which can be used for reference purposes when an individual rule is executed by Enterprise Architect during model validation.</p> <p>See <a href="#">Model Validation Example</a><sup>[160]</sup> for a detailed example of use of this method.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• CategoryId: String - should be passed the return value from the <a href="#">DefineRuleCategory</a><sup>[263]</sup> method.</li> <li>• ErrorType: EA.EnumMVErrorType - depending on the severity of the error being validated, can be: <ul style="list-style-type: none"> <li>• <b>mvErrorCritical</b></li> <li>• <b>mvError</b></li> <li>• <b>mvWarning</b>, or</li> <li>• <b>mvInformation</b>.</li> </ul> </li> <li>• ErrorMessage: String - contains an error string.</li> </ul>
<b>PutDiagramImageOnClipboard (string DiagramGUID, long Type)</b>	protected abstract: <i>Boolean</i>	<p>Copies an image of the specified diagram to the clipboard.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• DiagramGUID: String - the GUID (in XML format) of the diagram to copy.</li> <li>• Type: Long - the file type. <ul style="list-style-type: none"> <li>• If Type = <b>0</b> then it is a metafile</li> <li>• If Type = <b>1</b> then it is a Device Independent Bitmap.</li> </ul> </li> </ul>
<b>PutDiagramImageToFile (string Diagram GUID, string FileName, long Type)</b>	protected abstract: <i>Boolean</i>	<p>Saves an image of the specified diagram to file.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• DiagramGUID: String - the GUID (in XML format) of the diagram to save.</li> <li>• FileName: String - the name of the file to save the diagram into.</li> <li>• Type: Long - the file type. <ul style="list-style-type: none"> <li>• If type = <b>0</b> then it is a metafile</li> <li>• If type = <b>1</b> then it uses the file type from the name extension (that is, .bmp, .jpg, .gif, .png, .tga)</li> </ul> </li> </ul>
<b>ReloadProject ()</b>	protected abstract: <i>Boolean</i>	<p>Reloads the current project. This is a convenient method to refresh the current loaded project (in case of outside changes to the .EAP file).</p>
<b>RunReport (string PackageGUID, string TemplateName, string Filename)</b>	protected abstract: <i>Void</i>	<p>Runs a named RTF report.</p> <p>Parameters:</p> <ul style="list-style-type: none"> <li>• PackageGUID: String - the GUID (in XML format) of the package to run the report on.</li> <li>• TemplateName: String - the RTF report template to use. If the PackageGUID has a stereotype of <i>MasterDocument</i>, the template is not required.</li> <li>• FileName: String - the file name to store the generated report in.</li> </ul>
<b>RunHTMLReport (string PackageGUID, string</b>	<i>String</i>	<p>Runs an HTML report (same as <b>Documentation   HTML Documentation</b> when you right-click on a</p>

Method	Type	Notes
<b>ExportPath</b> , string <b>ImageFormat</b> , string <b>Style</b> , string <b>Extension</b> , string		package in the <b>Project Browser</b> ). Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to run the report on.</li> <li>ExportPath: String - the file name to store the generated report in.</li> <li>ImageFormat: String.</li> <li>Style: String.</li> <li>Extension: String.</li> </ul>
<b>SaveControlledPackage</b> (string PackageGUID)	String	Saves a package that has been configured as a controlled package, to XML. Only the package GUID is required, Enterprise Architect picks the rest up from the package control information. Parameter: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package to save.</li> </ul>
<b>SaveDiagramImageToFile</b> (string Filename)	protected abstract: String	Saves a diagram image of the current diagram to file. Parameters: <ul style="list-style-type: none"> <li>FileName: String - the filename of the image to save.</li> </ul>
<b>ShowWindow</b> (long Show)	protected abstract: Void	Shows or hides the Enterprise Architect User Interface. Parameters: <ul style="list-style-type: none"> <li>Show: Long.</li> </ul>
<b>SynchronizeClass</b> (string ElementGUID, string ExtraOptions)	Boolean	Synchronizes a Class with the latest source code. Parameters: <ul style="list-style-type: none"> <li>ElementGUID: String - the GUID (in XML format) of the element to update from code.</li> <li>ExtraOptions: String - enables extra options to be given to the command; currently unused.</li> </ul>
<b>SynchronizePackage</b> (string PackageGUID, string ExtraOptions)	Boolean	Synchronizes each Class in a package with the latest source code. Parameters: <ul style="list-style-type: none"> <li>PackageGUID: String - the GUID (in XML format) of the package containing the elements to update from code.</li> <li>ExtraOptions: String - enables extra options to be given to the command; currently enables synchronization of all child packages (children) - for example: <i>children=1</i>.</li> </ul>
<b>TransformElement</b> (string TransformName, string ElementGUID, string TargetPackage, string ExtraOptions)	Boolean	Transforms an element into a package. Parameters: <ul style="list-style-type: none"> <li>TransformName: String - specifies the transformation that should be executed.</li> <li>ElementGUID: String - the GUID (in XML format) of the element to transform.</li> <li>TargetPackageGUID: String - the GUID (in XML format) of the package to transform into.</li> <li>ExtraOptions: String - enables extra options to be</li> </ul>



Method	Type	Notes
		given to the command; currently unused.
<b>TransformPackage (string TransformName, string SourcePackage, string TargetPackage, string ExtraOptions)</b>	<i>Boolean</i>	Runs a transformation on the contents of a package. Parameters: <ul style="list-style-type: none"> <li>• TransformName: String - specifies the transformation that should be executed.</li> <li>• SourcePackageGUID: String - the GUID (in XML format) of the package to transform.</li> <li>• TargetPackageGUID: String - the GUID (in XML format) of the package to transform into.</li> <li>• ExtraOptions: String - enables extra options to be given to the command; currently unused.</li> </ul>
<b>XMLtoGUID (string GUID)</b>	<i>String</i>	Changes a GUID in XML format to the form used inside Enterprise Architect. Parameters: <ul style="list-style-type: none"> <li>• GUID: String - the XML style GUID to convert to Enterprise Architect internal format.</li> </ul>

## 7.2.10 Code Samples

This topic contains various code examples indicating how to use the Automation Interface, written in VB.Net:

- [Open the Repository](#) <sup>[273]</sup>
- [Iterate Through a .EAP File](#) <sup>[274]</sup>
- [Add and Manage Packages](#) <sup>[274]</sup>
- [Add and Manage Elements](#) <sup>[275]</sup>
- [Add a Connector](#) <sup>[275]</sup>
- [Add and Manage Diagrams](#) <sup>[276]</sup>
- [Add and Delete Features](#) <sup>[277]</sup>
- [Element Extras](#) <sup>[277]</sup>
- [Repository Extras](#) <sup>[280]</sup>
- [Stereotypes](#) <sup>[281]</sup>
- [Work with Attributes](#) <sup>[282]</sup>
- [Work with Methods](#) <sup>[282]</sup>

### 7.2.10.1 Open the Repository

#### public Object

"An example of how to open an Enterprise Architect repository  
"in VB.Net.

```
Public Class AutomationExample
```

```
"class level variable for Repository
Public m_Repository As Object
```

```
Public Sub Run()
```

```
try
"create the repository object
m_Repository = CreateObject("EA.Repository")
```

```
"open an EAP file
m_Repository.OpenFile("F:\Test\EAAuto.EAP")
"use the Repository in any way required
'DumpModel
```

```

    "close the repository and tidy up
    m_Repository.Exit()
    m_Repository = Nothing

....catch e as exception
    Console.WriteLine(e)
End try
End Sub
end Class

```

### 7.2.10.2 Iterate Through a .EAP File

#### public Object

```

"Assume repository has already been opened.

"Start at the model level
Sub DumpModel()
    Dim idx as Integer
    For idx=0 to m_Repository.Models.Count-1
        DumpPackage("",m_Repository.Models.GetAt(idx))
    Next
End Sub

'output package name, then element contents, then process child packages
Sub DumpPackage(Indent as String, Package as Object)
    Dim idx as Integer
    Console.WriteLine(Indent + Package.Name)
    DumpElements(Indent + " ", Package)

    For idx = 0 to Package.Packages.Count-1
        DumpPackage(Indent + " ", Package.Packages.GetAt(idx))
    Next
End Sub

"dump element name
Sub DumpElements(Indent as String, Package as Object)
    Dim idx as Integer
    For idx = 0 to Package.Elements.Count-1
        Console.WriteLine(Indent + "::" + Package.Elements.GetAt(idx).Name)
    Next
End Sub

```

### 7.2.10.3 Add and Manage Packages

#### public Object

Example illustrating how to add a Model or a Package.

```

Sub TestPackageLifecycle

    Dim idx as integer
    Dim idx2 as integer
    Dim package as object
    Dim model as object
    Dim o as object

    "first add a new Model

    model = m_Repository.Models.AddNew("AdvancedModel", "")
    If not model.Update() Then
        Console.WriteLine(model.GetLastError())
    End If

    "refresh the models collection
    m_Repository.Models.Refresh

    "now work through models collection and add a package

```

```

For idx = 0 to m_Repository.Models.Count - 1
  o = m_Repository.Models.GetAt(idx)
  Console.WriteLine(o.Name)
  If o.Name = "AdvancedModel" Then
    package = o.Packages.Addnew("Subpackage", "Nothing")
    If not package.Update() Then
      Console.WriteLine(package.GetLastError())
    End If

    package.Element.Stereotype = "system"
    package.Update

    "for testing purposes just delete the
    "newly created Model and its contents
    m_Repository.Models.Delete(idx)

  End If
Next

End Sub

```

#### 7.2.10.4 Add and Manage Elements

##### public Object

```

"Add and delete elements in a package.

Sub ElementLifecycle

  Dim package as Object
  Dim element as Object

  package = m_Repository.GetPackageByID(2)
  element = package.elements.AddNew("Login to Website", "UseCase")
  element.Stereotype = "testcase"
  element.Update
  package.elements.Refresh()

  Dim idx as integer

  "note the repeated calls to "package.elements.GetAt"
  "in general you should make this call once and assign to a local
  "variable - in the example below, Enterprise Architect loads the element required
  "everytime a call is made - rather than loading once and keeping
  "a local reference

  For idx = 0 to package.elements.count-1
    Console.WriteLine(package.elements.GetAt(idx).Name)
    If (package.elements.GetAt(idx).Name = "Login to Website" and _
        package.elements.GetAt(idx).Type = "UseCase") Then
      package.elements.deleteat(idx, false)
    End If
  Next
End Sub

```

#### 7.2.10.5 Add a Connector

##### public Object

```

"Add a connector and set values.

Sub ConnectorTest

  Dim source as object
  Dim target as object
  Dim con as object
  Dim o as object

```

```

Dim client as object
Dim supplier as object

"use ElementID's to quickly load an element in this example
"... you must find suitable ID's in your model

source = m_Repository.GetElementByID(129)
target = m_Repository.GetElementByID(169)

con = source.Connectors.AddNew ("test link 2", "Association")

"again- replace ID with a suitable one from your model
con.SupplierID = 169

If not con.Update Then
    Console.WriteLine(con.GetLastError)
End If
source.Connectors.Refresh

Console.WriteLine("Connector Created")

o = con.Constraints.AddNew ("constraint2","type")
If not o.Update Then
    Console.WriteLine(o.GetLastError)
End If

o = con.TaggedValues.AddNew ("Tag","Value")
If not o.Update Then
    Console.WriteLine(o.GetLastError)
End If

"use the client and supplier ends to set
"additional information

client = con.ClientEnd
client.Visibility = "Private"
client.Role = "m_client"
client.Update
supplier = con.SupplierEnd
supplier.Visibility = "Protected"
supplier.Role = "m_supplier"
supplier.Update

Console.WriteLine("Client and Supplier set")

Console.WriteLine(client.Role)
Console.WriteLine(supplier.Role)

End Sub

```

### 7.2.10.6 Add and Manage Diagrams

#### public Object

"An example of how to create a diagram and add an element to it.  
 "Note the optional use of element rectangle setting using  
 "left,right,top and bottom dimensions in AddNew call.

```

Sub DiagramLifeCycle

    Dim diagram as object
    Dim v as object
    Dim o as object
    Dim package as object

    Dim idx as Integer
    Dim idx2 as integer

    package = m_Repository.GetPackageByID(5)

    diagram = package.Diagrams.AddNew("Logical Diagram","Logical")
    If not diagram.Update Then
        Console.WriteLine(diagram.GetLastError)
    End If
End Sub

```

```

End if

diagram.Notes = "Hello there this is a test"
diagram.update()

o = package.Elements.AddNew("ReferenceType","Class")
o.Update

" add element to diagram - supply optional rectangle co-ordinates

v = diagram.DiagramObjects.AddNew("l=200;r=400;t=200;b=600;","")
v.ElementID = o.ElementID
v.Update

Console.WriteLine(diagram.DiagramID)

End Sub

```

### 7.2.10.7 Add and Delete Features

#### public Object

```

Dim element as object
Dim idx as integer
Dim attribute as object
Dim method as object

'just load an element by ID - you must
'substitute a valid ID from your model
element = m_Repository.GetElementByID(246)

"create a new method
method = element.Methods.AddNew("newMethod", "int")
method.Update
element.Methods.Refresh

'now loop through methods for Element - and delete our addition
For idx = 0 to element.Methods.Count-1
    method =element.Methods.GetAt(idx)
    Console.WriteLine(method.Name)
    If(method.Name = "newMethod") Then
        element.Methods.Delete(idx)
    End if
Next

'create an attribute
attribute = element.attributes.AddNew("NewAttribute", "int")
attribute.Update
element.attributes.Refresh

'loop through and delete our new attribute
For idx = 0 to element.attributes.Count-1
    attribute =element.attributes.GetAt(idx)
    Console.WriteLine(attribute.Name)
    If(attribute.Name = "NewAttribute") Then
        element.attributes.Delete(idx)
    End If
Next

```

### 7.2.10.8 Element Extras

#### public Object

"Examples of how to access and use element extras, such as scenarios, constraints and requirements.

```

Sub ElementExtras

    Dim element as object
    Dim o as object

```

```

Dim idx as Integer
Dim bDel as boolean
bDel = true

try
  element = m_Repository.GetElementByID(129)

  'manage constraints for an element
  'demonstrate addnew and delete
  o = element.Constraints.AddNew("Appended","Type")
  If not o.Update Then
    Console.WriteLine("Constraint error:" + o.GetLastError())
  End if
  element.Constraints.Refresh
  For idx = 0 to element.Constraints.Count -1
    o = element.Constraints.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
      If bDel Then element.Constraints.Delete (idx)
    End if
  Next

  'efforts
  o = element.Efforts.AddNew("Appended","Type")
  If not o.Update Then
    Console.WriteLine("Efforts error:" + o.GetLastError())
  End if
  element.Efforts.Refresh
  For idx = 0 to element.Efforts.Count -1
    o = element.Efforts.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
      If bDel Then element.Efforts.Delete (idx)
    End if
  Next

  'Risks
  o = element.Risks.AddNew("Appended","Type")
  If not o.Update Then
    Console.WriteLine("Risks error:" + o.GetLastError())
  End if
  element.Risks.Refresh
  For idx = 0 to element.Risks.Count -1
    o = element.Risks.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
      If bDel Then element.Risks.Delete (idx)
    End if
  Next

  'Metrics
  o = element.Metrics.AddNew("Appended","Change")
  If not o.Update Then
    Console.WriteLine("Metrics error:" + o.GetLastError())
  End if
  element.Metrics.Refresh
  For idx = 0 to element.Metrics.Count -1
    o = element.Metrics.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
      If bDel Then element.Metrics.Delete (idx)
    End if
  Next

  'TaggedValues
  o = element.TaggedValues.AddNew("Appended","Change")
  If not o.Update Then
    Console.WriteLine("TaggedValues error:" + o.GetLastError())
  End if
  element.TaggedValues.Refresh
  For idx = 0 to element.TaggedValues.Count -1
    o = element.TaggedValues.GetAt(idx)

```

```
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
        If bDel Then element.TaggedValues.Delete (idx)
    End if
Next

'Scenarios
o = element.Scenarios.AddNew("Appended","Change")
If not o.Update Then
    Console.WriteLine("Scenarios error:" + o.GetLastError())
End if
element.Scenarios.Refresh
For idx = 0 to element.Scenarios.Count -1
    o = element.Scenarios.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Appended") Then
        If bDel Then element.Scenarios.Delete (idx)
    End if
Next

'Files
o = element.Files.AddNew("MyFile","doc")
If not o.Update Then
    Console.WriteLine("Files error:" + o.GetLastError())
End if
element.Files.Refresh
For idx = 0 to element.Files.Count -1
    o = element.Files.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="MyFile") Then
        If bDel Then element.Files.Delete (idx)
    End if
Next

'Tests
o = element.Tests.AddNew("TestPlan","Load")
If not o.Update Then
    Console.WriteLine("Tests error:" + o.GetLastError())
End if
element.Tests.Refresh
For idx = 0 to element.Tests.Count -1
    o = element.Tests.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="TestPlan") Then
        If bDel Then element.Tests.Delete (idx)
    End if
Next

'Defect
o = element.Issues.AddNew("Broken","Defect")
If not o.Update Then
    Console.WriteLine("Issues error:" + o.GetLastError())
End if
element.Issues.Refresh
For idx = 0 to element.Issues.Count -1
    o = element.Issues.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Broken") Then
        If bDel Then element.Issues.Delete (idx)
    End if
Next

'Change
o = element.Issues.AddNew("Change","Change")
If not o.Update Then
    Console.WriteLine("Issues error:" + o.GetLastError())
End if
element.Issues.Refresh
For idx = 0 to element.Issues.Count -1
    o = element.Issues.GetAt(idx)
    Console.WriteLine(o.Name)
    If(o.Name="Change") Then
        If bDel Then element.Issues.Delete (idx)
    End if
End if
```

```

Next

    catch e as exception
        Console.WriteLine(element.Methods.GetLastError())
        Console.WriteLine(e)
    End try

End Sub

```

### 7.2.10.9 Repository Extras

#### public Object

```

" Examples of how to access repository
" collections for system level information.

Sub RepositoryExtras

    Dim o as object
    Dim idx as integer

    'issues
    o = m_Repository.Issues.AddNew("Problem","Type")
    If(o.Update=false) Then
        Console.WriteLine (o.GetLastError())
    End if
    o = nothing
    m_Repository.Issues.Refresh
    For idx = 0 to m_Repository.Issues.Count-1
        Console.WriteLine(m_Repository.Issues.GetAt(idx).Name)
        If(m_Repository.Issues.GetAt(idx).Name = "Problem") then
            m_Repository.Issues.DeleteAt(idx,false)
            Console.WriteLine("Delete Issues")
        End if
    Next

    'tasks
    o = m_Repository.Tasks.AddNew("Task 1","Task type")
    If(o.Update=false) Then
        Console.WriteLine ("error - " + o.GetLastError())
    End if
    o = nothing
    m_Repository.Tasks.Refresh
    For idx = 0 to m_Repository.Tasks.Count-1
        Console.WriteLine(m_Repository.Tasks.GetAt(idx).Name)
        If(m_Repository.Tasks.GetAt(idx).Name = "Task 1") then
            m_Repository.Tasks.DeleteAt(idx,false)
            Console.WriteLine("Delete Tasks")
        End if
    Next

    'glossary
    o = m_Repository.Terms.AddNew("Term 1","business")
    If(o.Update=false) Then
        Console.WriteLine ("error - " + o.GetLastError())
    End if
    o = nothing
    m_Repository.Terms.Refresh
    For idx = 0 to m_Repository.Terms.Count-1
        Console.WriteLine(m_Repository.Terms.GetAt(idx).Term)
        If(m_Repository.Terms.GetAt(idx).Term = "Term 1") then
            m_Repository.Terms.DeleteAt(idx,false)
            Console.WriteLine("Delete Terms")
        End if
    Next

    'authors
    o = m_Repository.Authors.AddNew("Joe B","Writer")
    If(o.Update=false) Then
        Console.WriteLine (o.GetLastError())
    End if
    o = nothing

```



```

m_Repository.Authors.Refresh
For idx = 0 to m_Repository.authors.Count-1
  COnsole.WriteLine(m_Repository.Authors.GetAt(idx).Name)
  If(m_Repository.authors.GetAt(idx).Name = "Joe B") then
    m_Repository.authors.DeleteAt(idx,false)
    Console.WriteLine("Delete Authors")
  End if
Next

o = m_Repository.Clients.AddNew("Joe Sphere","Client")
If(o.Update=false) Then
  Console.WriteLine (o.GetLastError())
End if
o = nothing
m_Repository.Clients.Refresh
For idx = 0 to m_Repository.Clients.Count-1
  COnsole.WriteLine(m_Repository.Clients.GetAt(idx).Name)
  If(m_Repository.Clients.GetAt(idx).Name = "Joe Sphere") then
    m_Repository.Clients.DeleteAt(idx,false)
    Console.WriteLine("Delete Clients")
  End if
Next

o = m_Repository.Resources.AddNew("Joe Worker","Resource")
If(o.Update=false) Then
  Console.WriteLine (o.GetLastError())
End if
o = nothing
m_Repository.Resources.Refresh
For idx = 0 to m_Repository.Resources.Count-1
  COnsole.WriteLine(m_Repository.Resources.GetAt(idx).Name)
  If(m_Repository.Resources.GetAt(idx).Name = "Joe Worker") then
    m_Repository.Resources.DeleteAt(idx,false)
    Console.WriteLine("Delete Resources")
  End if
Next

End Sub

```

### 7.2.10.10 Stereotypes

#### public Object

```

Sub TestStereotypes

  Dim o as object
  Dim idx as integer

  "add a new stereotype to the Stereotypes collection
  o = m_Repository.Stereotypes.AddNew("funky","class")
  If(o.Update=false) Then
    Console.WriteLine (o.GetLastError())
  End if
  o = nothing

  "make sure you refresh
  m_Repository.Stereotypes.Refresh

  "then iterate through - deleting our new entry in the process
  For idx = 0 to m_Repository.Stereotypes.Count-1
    COnsole.WriteLine(m_Repository.Stereotypes.GetAt(idx).Name)
    If(m_Repository.Stereotypes.GetAt(idx).Name = "funky") then
      m_Repository.Stereotypes.DeleteAt(idx,false)
      Console.WriteLine("Delete element")
    End if
  Next

End Sub

```

### 7.2.10.11 Work with Attributes

#### public Object

"An example of working with attributes.

Sub AttributeLifecycle

```

Dim element as object
Dim o as object
Dim t as object
Dim idx as Integer
Dim idx2 as integer
try
    element = m_Repository.GetElementByID(129)

    For idx = 0 to element.Attributes.Count -1

        Console.WriteLine("attribute=" + element.Attributes.GetAt(idx).Name)

        o = element.Attributes.GetAt(idx)
        t = o.Constraints.AddNew("> 123", "Precision")
        t.Update()
        o.Constraints.Refresh
        For idx2 = 0 to o.Constraints.Count-1
            t = o.Constraints.GetAt(idx2)
            Console.WriteLine("Constraint: " + t.Name)
            If(t.Name="> 123") Then
                o.Constraints.DeleteAt(idx2, false)
            End if
        Next

        For idx2 = 0 to o.TaggedValues.Count-1
            t = o.TaggedValues.GetAt(idx2)
            If(t.Name = "Type2") Then
                Console.WriteLine("deleteing")
                o.TaggedValues.DeleteAt(idx2, true)
            End if
        Next

        t = o.TaggedValues.AddNew("Type2", "Number")
        t.Update
        o.TaggedValues.Refresh
        For idx2 = 0 to o.TaggedValues.Count-1
            t = o.TaggedValues.GetAt(idx2)
            Console.WriteLine("Tagged Value: " + t.Name)
        Next

        If(element.Attributes.GetAt(idx).Name = "m_Tootle") Then
            Console.WriteLine("delete attribute")
            element.Attributes.DeleteAt(idx, false)
        End If

    Next

    catch e as exception
        Console.WriteLine(element.Attributes.GetLastError())
        Console.WriteLine(e)
    End try
End Sub

```

### 7.2.10.12 Work with Methods

#### public Object

"An example of working with the Methods collection of an element - and with Method collections.

Sub MethodLifeCycle

```

Dim element as object
Dim method as object

```

```
Dim t as object
Dim idx as Integer
Dim idx2 as integer

try
  element = m_Repository.GetElementByID(129)

  For idx = 0 to element.Methods.Count - 1
    method = element.Methods.GetAt(idx)
    Console.WriteLine(method.Name)

    t = method.PreConditions.AddNew("TestConstraint", "something")
    If t.Update = false Then
      Console.WriteLine("PreConditions: " + t.GetLastError)
    End if

    method.PreConditions.Refresh
    For idx2 = 0 to method.PreConditions.Count-1
      t = method.PreConditions.GetAt(idx2)
      Console.WriteLine("PreConditions: " + t.Name)
      If t.Name = "TestConstraint" Then
        method.PreConditions.DeleteAt(idx2, false)
      End If
    Next

    t = method.PostConditions.AddNew("TestConstraint", "something")
    If t.Update = false Then
      Console.WriteLine("PostConditions: " + t.GetLastError)
    End if

    method.PostConditions.Refresh
    For idx2 = 0 to method.PostConditions.Count-1
      t = method.PostConditions.GetAt(idx2)
      Console.WriteLine("PostConditions: " + t.Name)
      If t.Name = "TestConstraint" Then
        method.PostConditions.DeleteAt(idx2, false)
      End If
    Next

    t = method.TaggedValues.AddNew("TestTaggedValue", "something")
    If t.Update = false Then
      Console.WriteLine("Tagged Values: " + t.GetLastError)
    End if

    For idx2 = 0 to method.TaggedValues.Count-1
      t = method.TaggedValues.GetAt(idx2)
      Console.WriteLine("Tagged Value: " + t.Name)
      If(t.Name= "TestTaggedValue") Then
        method.TaggedValues.DeleteAt(idx2, false)
      End If
    Next

    t = method.Parameters.AddNew("TestParam", "string")
    If t.Update = false Then
      Console.WriteLine("Parameters: " + t.GetLastError)
    End if

    method.Parameters.Refresh
    For idx2 = 0 to method.Parameters.Count-1
      t = method.Parameters.GetAt(idx2)
      Console.WriteLine("Parameter: " + t.Name)
      If(t.Name="TestParam") Then
        method.Parameters.DeleteAt(idx2, false)
      End If
    Next

    method = nothing
  Next
catch e as exception
  Console.WriteLine(element.Methods.GetLastError())
  Console.WriteLine(e)
End try

End Sub
```

# Index

## - A -

### Add

- Code Modules In MDG Technology Wizard 38
- Diagram Type In MDG Technology Wizard 34
- Enumeration Tags To Stereotypes 13
- Images In MDG Technology Wizard 41
- Linked Document Template In MDG Technology Wizard 43
- MDA Transforms In MDG Technology Wizard 40
- Pattern In MDG Technology Wizard 33
- Profile In MDG Technology Wizard 33
- RTF Report Template In MDG Technology Wizard 42
- Scripts In MDG Technology Wizard 41
- Shape Script To Stereotype In Profile 14
- Tagged Value Types In MDG Technology Wizard 37
- Task Panel In MDG Technology Wizard 36
- Toolbox In MDG Technology Wizard 35
- Add And Delete Attributes
  - Automation Interface Code Example 277
- Add And Delete Methods
  - Automation Interface Code Example 277
- Add And Manage Diagrams
  - Automation Interface Code Example 276
- Add And Manage Elements
  - Automation Interface Code Example 275
- Add And Manage Packages
  - Automation Interface Code Example 274
- Add Connector
  - Automation Interface Code Example 275
- Add Stereotypes
  - Automation Interface Code Example 281
- Add-In
  - And Enterprise ArchitectDeadlocks (.NET) 124
  - COM Interoperability 124
  - Concurrent Method Calls 124
  - Create 122
  - Create, Define Menu Items 122
  - Deploy 123
  - Disable 126
  - Enable 126
  - Events 127
  - Holding State Information 124
  - Manage 126
  - Manager 126

- Pre-2004 124
- Re-entrancy 124
- Run Functions From Tasks Pane 54
- Search 126
- Search Data 127
- Tasks 122
- Visual Basic Issues 124

### Add-In Event

- EA\_Connect 128
- EA\_Disconnect 128
- EA\_GetMenuItems 129
- EA\_GetmenuState 129
- EA\_MenuClick 130
- EA\_OnOutputItemClicked 131
- EA\_OnOutputItemDoubleClicked 131
- EA\_ShowHelp 132

### Add-In Model

- Add-In Event, EA\_Connect 128
- Add-In Event, EA\_Disconnect 128
- Add-In Event, EA\_GetMenuItems 129
- Add-In Event, EA\_GetMenuState 129
- Add-In Event, EA\_MenuClick 130
- Add-In Event, EA\_OnOutputItemClicked 131
- Add-In Event, EA\_OnOutputItemDoubleClicked 131
- Add-In Event, EA\_ShowHelp 132
- Add-In Event, Overview 127
- Add-In Tasks 122
- Benefits 121
- Broadcast Event, EA\_FileClose 134
- Broadcast Event, EA\_FileNew 134
- Broadcast Event, EA\_FileOpen 133
- Broadcast Event, EA\_OnPostInitialized 145
- Broadcast Event, EA\_OnPostTransform 146
- Broadcast Event, EA\_OnPreExitInstance 141
- Broadcast Event, EA\_OnRetrieveModelTemplate 163
- Broadcast Events 133
- Compartment Events 153
- Compartment Events, EA\_GetCompartmentData 153
- Compartment Events, EA\_QueryAvailableCompartments 153
- Context Item Events 150
- Context Item Events, EA\_OnContextItemChanged 150, 152
- Context Item Events, EA\_OnContextItemDoubleClicked 151
- Create Add-In 122
- Create Add-In, Tricks and Traps 124
- Create Custom View 164
- Custom View 164
- EA\_Connect 128

- Add-In Model
- EA\_Disconnect 128
  - EA\_FileClose 134
  - EA\_FileNew 134
  - EA\_FileOpen 133
  - EA\_GetCompartmentData 153
  - EA\_GetMenuItems 129
  - EA\_GetMenuState 129
  - EA\_MenuClick 130
  - EA\_OnContextItemChanged 150, 152
  - EA\_OnContextItemDoubleClicked 151
  - EA\_OnDeleteTechnology 149
  - EA\_OnEndValidation 156
  - EA\_OnImportTechnology 149
  - EA\_OnInitialize\_Technologies 147
  - EA\_OnInitializeUserRules 155
  - EA\_OnOutputItemClicked 131
  - EA\_OnOutputItemDoubleClicked 131
  - EA\_OnPostActivateTechnology 148
  - EA\_OnPostInitialized 145
  - EA\_OnPostNewAttribute 144
  - EA\_OnPostNewConnector 142
  - EA\_OnPostNewDiagramObject 143
  - EA\_OnPostNewElement 142
  - EA\_OnPostNewMethod 144
  - EA\_OnPostNewPackage 145
  - EA\_OnPostTransform 146
  - EA\_OnPreActivateTechnology 147
  - EA\_OnPreDeleteConnector 135
  - EA\_OnPreDeleteDiagram 136
  - EA\_OnPreDeleteElement 135
  - EA\_OnPreDeletePackage 136
  - EA\_OnPreDeleteTechnology 148
  - EA\_OnPreExitInstance 141
  - EA\_OnPreNewAttribute 139
  - EA\_OnPreNewConnector 138
  - EA\_OnPreNewDiagramObject 139
  - EA\_OnPreNewElement 137
  - EA\_OnPreNewMethod 140
  - EA\_OnPreNewPackage 141
  - EA\_OnRetrieveModelTemplate 163
  - EA\_OnRunAttributeRule 158
  - EA\_OnRunConnectorRule 158
  - EA\_OnRunDiagramRule 157
  - EA\_OnRunElementRule 156
  - EA\_OnRunMethodRule 159
  - EA\_OnRunPackageRule 157
  - EA\_OnRunParameterRule 159
  - EA\_OnStartValidation 156
  - EA\_QueryAvailableCompartments 153
  - EA\_ShowHelp 132
  - Interface 121
  - Introduction 121
  - MDG Add-Ins 165
  - MDG Add-Ins, MDG Events 165
  - MDG Events, MDG\_BuildProject 166
  - MDG Events, MDG\_Connect 166
  - MDG Events, MDG\_Disconnect 167
  - MDG Events, MDG\_GetConnectedPackages 167
  - MDG Events, MDG\_GetProperty 168
  - MDG Events, MDG\_Merge 169
  - MDG Events, MDG\_NewClass 170
  - MDG Events, MDG\_PostGenerate 171
  - MDG Events, MDG\_PostMerge 171
  - MDG Events, MDG\_PreGenerate 172
  - MDG Events, MDG\_PreMerge 172
  - MDG Events, MDG\_PreReverse 173
  - MDG Events, MDG\_RunExe 174
  - MDG Events, MDG\_View 174
  - Model Validation Broadcasts 155
  - Model Validation Broadcasts, EA\_OnEndValidation 156
  - Model Validation Broadcasts, EA\_OnInitializeUserRules 155
  - Model Validation Broadcasts, EA\_OnRunAttributeRule 158
  - Model Validation Broadcasts, EA\_OnRunConnectorRule 158
  - Model Validation Broadcasts, EA\_OnRunDiagramRule 157
  - Model Validation Broadcasts, EA\_OnRunElementRule 156
  - Model Validation Broadcasts, EA\_OnRunMethodRule 159
  - Model Validation Broadcasts, EA\_OnRunPackageRule 157
  - Model Validation Broadcasts, EA\_OnRunParameterRule 159
  - Model Validation Broadcasts, EA\_OnStartValidation 156
  - Model Validation Example 160
  - Post-New Events 142
  - Post-New Events, EA\_OnPostNewAttribute 144
  - Post-New Events, EA\_OnPostNewConnector 142
  - Post-New Events, EA\_OnPostNewDiagramObject 143
  - Post-New Events, EA\_OnPostNewElement 142
  - Post-New Events, EA\_OnPostNewMethod 144
  - Post-New Events, EA\_OnPostNewPackage 145
  - Pre-Deletion Events 135
  - Pre-Deletion Events, EA\_OnPreDeleteConnector 135

- Add-In Model
  - Pre-Deletion Events, EA\_OnPreDeleteDiagram 136
  - Pre-Deletion Events, EA\_OnPreDeleteElement 135
  - Pre-Deletion Events, EA\_OnPreDeletePackage 136
  - Pre-New Events 137
  - Pre-New Events, EA\_OnPreNewAttribute 139
  - Pre-New Events, EA\_OnPreNewConnector 138
  - Pre-New Events, EA\_OnPreNewDiagramObject 139
  - Pre-New Events, EA\_OnPreNewElement 137
  - Pre-New Events, EA\_OnPreNewMethod 140
  - Pre-New Events, EA\_OnPreNewPackage 141
  - Search Data, XML Format 127
  - Technology Event, EA\_OnInitialize\_Technologies 147
  - Technology Events 146
  - Technology Events, EA\_OnDeleteTechnology 149
  - Technology Events, EA\_OnImportTechnology 149
  - Technology Events, EA\_OnPostActivateTechnology 148
  - Technology Events, EA\_OnPreActivateTechnology 147
  - Technology Events, EA\_OnPreDeleteTechnology 148
- App Object
  - Automation Interface 185
- Attribute
  - Add And Delete, Automation Interface Code Example 277
  - Automation Interface, ElementFeatures Package 235
  - Of Toolbox Page 46
  - PData & StyleEx, Diagram Profiles 51
  - Supported, Create Composite Elements 21
  - Supported, Define Child Diagram Types 21
  - Supported, Metatype In UML Profiles 18
  - Supported, Stereotype In UML Profiles 18
  - Work With, Automation Interface Code Example 282
- AttributeConstraint
  - Automation Interface, ElementFeatures Package 237
- AttributeTag
  - Automation Interface, ElementFeatures Package 238
- Author
  - Attributes 203
  - Methods 203
- Author Collection
- Automation Interface Repository 203
- Automation Interface
  - App Object 185
  - Attribute, ElementFeatures Package 235
  - AttributeConstraint, ElementFeatures Package 237
  - AttributeTag, ElementFeatures Package 238
  - Available Resources 181
  - Call Executables From Enterprise Architect 176
  - Call From Enterprise Architect 180
  - Code Example, Add And Delete Attributes 277
  - Code Example, Add And Delete Methods 277
  - Code Example, Add And Manage Diagrams 276
  - Code Example, Add And Manage Elements 275
  - Code Example, Add And Manage Packages 274
  - Code Example, Add Connector 275
  - Code Example, Add Stereotypes 281
  - Code Example, Iterate Through EAP File 274
  - Code Example, Open The Repository 273
  - Code Example, Use Element Extras 277
  - Code Example, Use Repository Extras 280
  - Code Example, Work With Attributes 282
  - Code Example, Work With Methods 282
  - Code Examples, Introduction 273
  - Connect From Borland Delphi 7.0 176
  - Connect From Java 176
  - Connect From MS C# 176
  - Connect From MS Visual Basic 6.0 176
  - Connect To 176
  - Connector Package Diagram 246
  - Connector, Connector Package 248
  - ConnectorConstraint, Connector Package 247
  - ConnectorEnd, Connector Package 251
  - ConnectorTag, Connector Package 252
  - ConstLayoutStyles Enum 186
  - Constraint, Element Package 220
  - CreateModelType Enum 186
  - CustomProperties Collection, ElementFeatures Package 239
  - Diagram Package 254
  - Diagram, Diagram Package 255
  - DiagramLinks, Diagram Package 258
  - DiagramObjects, Diagram Package 259
  - Effort, Element Package 220
  - Element Package Diagram 218
  - Element Package, File 228
  - Element, Element Package 221
  - ElementFeatures Package Diagram 235

- Automation Interface
  - EmbeddedElements Collection, ElementFeatures Package 239
  - Enumerations 186
  - EnumRelationSetType Enum 187
  - Examples 176
  - Examples and Tips 179
  - Introduction 176
  - Issue, Element Package 228
  - MDGMenu Enum 187
  - Method, ElementFeatures Package 240
  - MethodConstraint, ElementFeatures Package 242
  - MethodTag, ElementFeatures Package 243
  - Metric, Element Package 229
  - Model 182
  - ObjectType Enum 188
  - Package 182
  - Parameter, ElementFeatures Package 243
  - Partitions Collection, ElementFeatures Package 244
  - Project Interface 262
  - Project, Project Interface 262
  - Properties, ElementFeatures Package 245
  - Property ElementFeatures Package 245
  - PropType Enum 188
  - Reference 181
  - ReloadType Enum 189
  - Repository 190
  - Repository Package 189, 209
  - Repository, Author Collection 203
  - Repository, Client Collection 204
  - Repository, Collection Class 205
  - Repository, Datatype 206
  - Repository, EventProperties 207
  - Repository, EventProperty 208
  - Repository, ModelWatcher 208
  - Repository, ProjectIssues 212
  - Repository, ProjectResource 213
  - Repository, PropertyType 214
  - Repository, Reference 215
  - Repository, Stereotype 216
  - Repository, Task 216
  - Repository, Term 217
  - Requirement, Element Package 230
  - Resource, Element Package 231
  - Risk, Element Package 232
  - RoleTag, Connector Package 253
  - Scenario, Element Package 232
  - Set Up Visual Basic 178
  - Swimlane, Diagram Package 262
  - SwimlaneDef, Diagram Package 260
  - Swimlanes, Diagram Package 261
  - TaggedValue, Element Package 233
  - Test, Element Package 234
  - Transitions Collection, ElementFeatures Package 246
  - Using 176
  - VB GetObject Support 185
  - XMIType Enum 189
- Available Resources
  - Automation Interface 181
- B -**
- Behavioral Model Templates 107
- Branching Macros
  - Code Template Syntax 104
- Broadcast Event
  - Add-In Model 133
  - EA\_FileClose 134
  - EA\_FileNew 134
  - EA\_FileOpen 133
  - EA\_OnPostInitialized 145
  - EA\_OnPostTransform 146
  - EA\_OnPreExitInstance 141
  - EA\_OnRetrieveModelTemplate 163
- Built-In
  - Diagram Types 51
- C -**
- Call
  - Automation Interface From Enterprise Architect 180
- CLASSGUID
  - Add-In Hidden Field 126
- CLASSTYPE
  - Add-In Hidden Field 126
- Client Collection
  - Automation Interface Repository 204
- Code Module
  - Add To MDG Technology 38
- Code Template
  - Custom Templates, Create 117
  - Default Templates 118
  - Editor 117
  - Editor, Add New Stereotyped Templates 119
  - Editor, Create Templates For Custom Languages 120
  - Editor, In SDK 116
  - Export 117
  - Framework, In SDK 86
  - Import 117

- Code Template
  - Syntax, Introduction 86
  - Syntax, Literal Text 86
  - Syntax, Macros 87
  - Syntax, Template Substitution Macros 87
- Code Template Syntax
  - Variable Definitions 115
  - Variable References 115
  - Variables 115
- Collection Class
  - Automation Interface Repository 205
- Collections, EASL
  - Action 109
  - Behaviors 109
  - Classifier 109
  - Construct 109
  - Node 109
  - State 109
  - State Machine 109
  - Transition 109
  - Trigger 109
  - Vertex 109
- Color Query
  - Shape Scripts 70
- Compartment Events
  - Add-In Model 153
  - EA\_GetCompartmentData 153
  - EA\_QueryAvailableCompartments 153
- Composite Elements
  - Metaclass, Create With Supported Attributes 21
- Concurrent Method Calls
  - In Add-Ins 124
- Conditional Substitution
  - Field Substitution Macros, Code Template Syntax 88
- Connect
  - To Automation Interface 176
- Connector
  - Add, Automation Interface Code Example 275
  - Automation Interface, Connector Package 248
  - Shape Script Properties 70
  - Tagged Value, Use 10
- Connector Package
  - Connector, Automation Interface 248
  - ConnectorConstraint, Automation Interface 247
  - ConnectorEnd, Automation Interface 251
  - ConnectorTag, Automation Interface 252
  - RoleTag, Automation Interface 253
- Connector Package Diagram
  - Automation Interface 246
- ConnectorConstraint
  - Automation Interface, Connector Package 247
- ConnectorEnd
  - Automation Interface, Connector Package 251
- ConnectorTag
  - Automation Interface, Connector Package 252
- ConstLayoutStyles Enum
  - Automation Interface 186
- Constraint
  - Automation Interface, Element Package 220
  - Profile 11
  - Stereotype 11
- Context Item Events
  - Add-In Model 150
  - EA\_OnContextItemChanged 150, 152
  - EA\_OnContextItemDoubleClicked 151
- Control Macros
  - Code Template Syntax 104
- Create
  - Add-In 122
  - Custom Tagged Values 84
  - Custom View, Add-In Model 164
  - Hidden Submenu In Toolbox Profile 46
  - Masked Tagged Values 84
  - MDG Technologies 28
  - Profiles 5
  - Reference Data Tagged Values 83
  - Structured Tagged Values 81
  - Tasks Pane Profiles 52
  - Toolbox Profile For MDG Technology 45
  - UML Profiles 5
- CreateModelType Enum
  - Automation Interface 186
- CTF
  - In SDK 86
- Custom
  - Diagram Types 50
  - Stereotypes 3
- Custom Language
  - Create Templates For In Code Template Editor 120
- Custom Tagged Values
  - Create 84
- Custom View
  - Add-In Model 164
- CustomProperties Collection
  - Automation Interface, ElementFeatures Package 239

## - D -

Datatype



- Datatype
    - Automation Interface Repository 206
  - Default
    - Templates 118
  - Default Appearance
    - Set For Profile Stereotype Objects 16
  - Default Templates
    - Override in Code Template Editor 118
  - Default Toolbox
    - Override In Profile 47
  - Define
    - Stereotype As Metatype 19
    - Stereotype Constraints 11
    - Tasks Pane Contexts 55
    - Tasks Pane Toolbox 53
    - Validation Configuration For MDG Technology 56
  - Define Menu Items
    - Create Add-In 122
  - Deploy
    - Add-In 123
    - MDG Technology From Add-In 57
    - MDG Technology From File 57
  - Diagram
    - Add And Manage, Automation Interface Code Example 276
    - Automation Interface, Diagram Package 255
    - Define Child Type, Supported Attributes 21
    - Profile Attributes, PData and StyleEx 51
    - Profiles 50
    - Save Profile 17
    - Types, Built In 51
    - Types, Custom 50
  - Diagram Package
    - Automation Interface 254
    - Diagram, Automation Interface 255
    - DiagramLinks, Automation Interface 258
    - DiagramObjects, Automation Interface 259
    - Swimlane, Automation Interface 262
    - SwimlaneDef, Automation Interface 260, 261
  - Diagram Type
    - Add To MDG Technology 34
  - DiagramLinks
    - Automation Interface, Diagram Package 258
  - DiagramObjects
    - Automation Interface, Diagram Package 259
  - Direct Substitution
    - Field Substitution Macros, Code Template Syntax 88
  - Disable
    - Add-Ins 126
  - Display
    - Connector Properties, Shape Scripts 70
    - Element Properties, Shape Scripts 70
  - Drawing Methods
    - Shape Scripts 66
- E -**
- EA\_Connect
    - Add-In Event 128
  - EA\_Disconnect
    - Add-In Event 128
  - EA\_FileClose
    - Broadcast Events, Add-In Model 134
  - EA\_FileNew
    - Broadcast Events, Add-In Model 134
  - EA\_FileOpen
    - Broadcast Events, Add-In Model 133
  - EA\_GetCompartmentData
    - Compartment Events, Add-In Model 153
  - EA\_GetMenuItems
    - Add-In Event 129
  - EA\_GetMenuState
    - Add-In Event 129
  - EA\_MenuClick
    - Add-In Event 130
  - EA\_OnContextItemChanged
    - Context Item Events, Add-In Model 150
  - EA\_OnContextItemDoubleClicked
    - Context Item Events, Add-In Model 151
  - EA\_OnDeleteTechnology
    - Technology Events, Add-In Model 149
  - EA\_OnEndValidation
    - Model Validation Broadcasts, Add-In Model 156
  - EA\_OnImportTechnology
    - Technology Events, Add-In Model 149
  - EA\_OnInitialize\_Technologies
    - Technology Events, Add-In Model 147
  - EA\_OnInitializeUserRules
    - Model Validation Broadcasts, Add-In Model 155
  - EA\_OnNotifyContextItemModified
    - Context Item Events, Add-In Model 152
  - EA\_OnOutputItemClicked
    - Add-In Event 131
  - EA\_OnOutputItemDoubleClicked
    - Add-In Event 131
  - EA\_OnPostActivateTechnology
    - Technology Events, Add-In Model 148
  - EA\_OnPostInitialized
    - Broadcast Events, Add-In Model 145
  - EA\_OnPostNewAttribute

- EA\_OnPostNewAttribute
  - Post-New Events, Add-In Model 144
- EA\_OnPostNewConnector
  - Post-New Events, Add-In Model 142
- EA\_OnPostNewDiagramObject
  - Post-New Events, Add-In Model 143
- EA\_OnPostNewElement
  - Post-New Events, Add-In Model 142
- EA\_OnPostNewMethod
  - Post-New Events, Add-In Model 144
- EA\_OnPostNewPackage
  - Post-New Events, Add-In Model 145
- EA\_OnPostTransform
  - Broadcast Events, Add-In Model 146
- EA\_OnPreActivateTechnology
  - Technology Events, Add-In Model 147
- EA\_OnPreDeleteConnector
  - Pre-Deletion Events, Add-In Model 135
- EA\_OnPreDeleteDiagram
  - Pre-Deletion Events, Add-In Model 136
- EA\_OnPreDeleteElement
  - Pre-Deletion Events, Add-In Model 135
- EA\_OnPreDeletePackage
  - Pre-Deletion Events, Add-In Model 136
- EA\_OnPreDeleteTechnology
  - Technology Events, Add-In Model 148
- EA\_OnPreExitInstance
  - Broadcast Events, Add-In Model 141
- EA\_OnPreNewAttribute
  - Pre-New Events, Add-In Model 139
- EA\_OnPreNewConnector
  - Pre-New Events, Add-In Model 138
- EA\_OnPreNewDiagramObject
  - Pre-New Events, Add-In Model 139
- EA\_OnPreNewElement
  - Pre-New Events, Add-In Model 137
- EA\_OnPreNewMethod
  - Pre-New Events, Add-In Model 140
- EA\_OnPreNewPackage
  - Pre-New Events, Add-In Model 141
- EA\_OnRetrieveModelTemplate
  - Broadcast Events, Add-In Model 163
- EA\_OnRunAttributeRule
  - Model Validation Broadcasts, Add-In Model 158
- EA\_OnRunConnectorRule
  - Model Validation Broadcasts, Add-In Model 158
- EA\_OnRunDiagramRule
  - Model Validation Broadcasts, Add-In Model 157
- EA\_OnRunElementRule
  - Model Validation Broadcasts, Add-In Model 156
- EA\_OnRunMethodRule
  - Model Validation Broadcasts, Add-In Model 159
- EA\_OnRunPackageRule
  - Model Validation Broadcasts, Add-In Model 157
- EA\_OnRunParameterRule
  - Model Validation Broadcasts, Add-In Model 159
- EA\_OnStartValidation
  - Model Validation Broadcasts, Add-In Model 156
- EA\_QueryAvailableCompartments
  - Compartment Events, Add-In Model 153
- EA\_ShowHelp
  - Add-In Event 132
- EAP File
  - Iterate Through, Automation Interface Code Example 274
- EASL
  - Behavioral Model Templates 107
  - Code Generation Macros, Behavioral Model 107
  - Enterprise Architect Simulation Library 107
- EASL Collections
  - Action 109
  - Behavior 109
  - Classifier 109
  - Construct 109
  - Node 109
  - State 109
  - State Machine 109
  - Transition 109
  - Trigger 109
  - Vertex 109
- EASL Properties
  - Action 111
  - Argument 111
  - Behavior 111
  - Call Event 111
  - ChangeEvent 111
  - Classifier 111
  - Condition 111
  - Construct 111
  - Edge 111
  - EventObject 111
  - Instance 111
  - Parameter 111
  - Primitive 111
  - PropertyObject 111
  - SignalEvent 111

- EASL Properties
    - State 111
    - StateMachine 111
    - TimeEvent 111
    - Transition 111
    - Trigger 111
    - Vertex 111
  - EASL\_GET
    - Code Generation Macro, Behavioral Model 107
  - EASLList
    - Code Generation Macro, Behavioral Model 107
  - Effort
    - Attributes 220
    - Automation Interface, Element Package 220
    - Methods 220
  - Element
    - Add And Manage, Automation Interface Code Example 275
    - Add To Profile 6
    - Automation Interface, Element Package 221
    - Shape Script Properties 70
    - Use Extras, Automation Interface Code Example 277
  - Element Package, Automation Interface
    - Constraint 220
    - Diagram 218
    - Effort 220
    - Element 221
    - File 228
    - Issue 228
    - Metric 229
    - Requirement 230
    - Resource 231
    - Risk 232
    - Scenario 232
    - TaggedValue 233
    - Test 234
  - Element Templates
    - And Profiles 3
  - ElementFeatures Package, Automation Interface
    - Attribute 235
    - AttributeConstraint 237
    - AttributeTag 238
    - CustomProperties Collection 239
    - Diagram 235
    - EmbeddedElements Collection 239
    - Method 240
    - MethodConstraint 242
    - MethodTag 243
    - Parameter 243
    - Partitions Collection 244
    - Properties 245
    - Property 245
    - Transitions Collection 246
  - EmbeddedElements Collection
    - Automation Interface, ElementFeatures Package 239
  - Enable
    - Add-Ins 126
  - Enterprise Architect
    - Add-In Model 121
    - Object Model, Introduction 176
    - SDK, Introduction 2
  - Enterprise Architect Simulation Library
    - Behavioral Model Templates 107
    - EASL Code Generation 107
    - EASL\_GET Macro 107
    - EASLList Macro 107
  - Enumeration
    - Automation Interface 186
    - ConstLayoutStyles 186
    - CreateModelType 186
    - EnumRelationSetType 187
    - MDGMenus 187
    - ObjectType 188
    - PropType 188
    - ReloadType 189
    - XMIType 189
  - Enumeration Elements
    - Add To Profiles 13
  - EnumRelationSetType Enum
    - Automation Interface 187
  - EventProperties
    - Automation Interface Repository 207
  - EventProperty
    - Automation Interface Repository 208
  - Examples And Tips
    - Automation Interface 179
  - Export
    - Code Templates 117
    - Profile 16
    - UML Profile 16
  - Extend
    - UML Toolbox Connectors 49
    - UML Toolbox Elements 48
- F -
- Field Substitution Macros
    - Access Data from Attributes 88
    - Access Data from Classes 88
    - Access Data from Operations 88
    - Access Data from Packages 88

- Field Substitution Macros
    - Access Data from Parameters 88
    - Conditional Substitution 88
    - Direct Substitution 88
  - File
    - Element Package, Automation Interface 228
  - Function
    - Macros, Code Template Syntax 101
- H -**
- Hidden Submenu
    - Create In Toolbox Profile 46
- I -**
- Icons
    - For Toolbox Items, Assign 47
  - Image
    - Add To MDG Technology 41
  - Import
    - Code Templates 117
  - Instance
    - Define Behavior On Creation, Supported Attributes 20
  - Introduction
    - To Enterprise Architect SDK 2
    - To Quick Linker 23
    - To Shape Scripts 59
    - To Tagged Value Types 79
  - Issue (Defect)
    - Automation Interface, Element Package 228
  - Iterate Through EAP File
    - Automation Interface Code Example 274
- L -**
- Language
    - Custom, Create Templates For In Code Template Editor 120
  - Linked Document Template
    - Add To MDG Technology 43
  - List Macro
    - Code Template Syntax 104
- M -**
- Macro
    - Branching 104
    - Code Template Syntax 104
    - Control 104
  - Field Substitution, Code Template Syntax 88
  - Function, Code Template Syntax 101
  - List 104
  - PI 104
  - Synchronization 104
  - Tagged Value, Code Template Syntax 100
  - Template Substitution, Code Template Syntax 87
  - Macros
    - Behavioral Model 107
    - Code Template Syntax 87
    - EASL Code Generation 107
    - EASL\_GET 107
    - EASLList 107
  - Manage
    - Add-Ins 126
  - Masked Tagged Values
    - Create 84
  - MDA Transform
    - Add To MDG Technology 40
  - MDG Add-Ins
    - Add-In Model 165
    - MDG Events 165
    - MDG\_BuildProject 166
    - MDG\_Connect 166
    - MDG\_Disconnect 167
    - MDG\_GetConnectedPackages 167
    - MDG\_GetProperty 168
    - MDG\_Merge 169
    - MDG\_NewClass 170
    - MDG\_PostGenerate 171
    - MDG\_PostMerge 171
    - MDG\_PreGenerate 172
    - MDG\_PreMerge 172
    - MDG\_PreReverse 173
    - MDG\_Run\_Exe 174
    - MDG\_View 174
  - MDG Events
    - Add-In Model 165
    - MDG\_BuildProject 166
    - MDG\_Connect 166
    - MDG\_Disconnect 167
    - MDG\_GetConnectedPackages 167
    - MDG\_GetProperty 168
    - MDG\_Merge 169
    - MDG\_NewClass 170
    - MDG\_PostGenerate 171
    - MDG\_PostMerge 171
    - MDG\_PreGenerate 172
    - MDG\_PreMerge 172
    - MDG\_PreReverse 173
    - MDG\_Run\_Exe 174

- MDG Events
  - MDG\_View 174
- MDG Technology
  - Create 28
  - Create Toolbox Profile For 45
  - Define Tasks Pane Profile 52
  - Define Validation Configuration 56
  - Deploy From Add-In 57
  - Deploy From File 57
  - In SDK 28
  - Include Custom Diagram Types 50
  - Incorporate Model Template 57
- MDG Technology Wizard
  - Add Code Modules 38
  - Add Diagram Type To Technology 34
  - Add Images 41
  - Add Linked Document Template To Technology 43
  - Add MDA Transforms 40
  - Add Pattern To Technology 33
  - Add Profile To Technology 33
  - Add RTF Report Template To Technology 42
  - Add Scripts 41
  - Add Tagged Value Types 37
  - Add Task Panel To Technology 36
  - Add Toolbox To Technology 35
  - Create Technologies 28
- MDG\_BuildProject
  - Add-In Model 166
- MDG\_Connect
  - Add-In Model 166
- MDG\_Disconnect
  - Add-In Model 167
- MDG\_GetConnectedPackages
  - Add-In Model 167
- MDG\_GetProperty
  - Add-In Model 168
- MDG\_Merge
  - Add-In Model 169
- MDG\_NewClass
  - Add-In Model 170
- MDG\_PostGenerate
  - Add-In Model 171
- MDG\_PostMerge
  - Add-In Model 171
- MDG\_PreGenerate
  - Add-In Model 172
- MDG\_PreMerge
  - Add-In Model 172
- MDG\_PreReverse
  - Add-In Model 173
- MDG\_Run\_Exe
  - Add-In Model 174
- MDG\_View
  - Add-In Model 174
- MDGMenus Enum
  - Automation Interface 187
- Menu
  - Items, Define In Add-In 122
- Metaclass
  - Add To Profile 6
- Method
  - Add And Delete, Automation Interface Code Example 277
  - Automation Interface, ElementFeatures Package 240
  - Work With, Automation Interface Code Example 282
- MethodConstraint
  - Automation Interface, ElementFeatures Package 242
- MethodTag
  - Automation Interface, ElementFeatures Package 243
- Metric
  - Automation Interface, Element Package 229
- MiscData 221
- Model
  - Automation Interface 182
  - Templates, Incorporate In Technology 57
- Model Search
  - Access From Add-In 126
  - Define In MTS File 44
- Model Validation
  - Define Configuration For MDG Technology 56
- Model Validation Broadcasts
  - Add-In Model 155
  - EA\_OnEndValidation 156
  - EA\_OnInitializeUserRules 155
  - EA\_OnRunAttributeRule 158
  - EA\_OnRunConnectorRule 158
  - EA\_OnRunDiagramRule 157
  - EA\_OnRunElementRule 156
  - EA\_OnRunMethodRule 159
  - EA\_OnRunPackageRule 157
  - EA\_OnRunParameterRule 159
  - EA\_OnStartValidation 156
  - Model Validation Example 160
- Model Views
  - Define In MTS File 44
- Models Collection 190
- ModelWatcher
  - Automation Interface Repository 208
- MTS File
  - Advanced Options 44

- MTS File
    - Create 44
    - Incorporate Model Search 44
    - Incorporate Model View 44
    - Working With 44
  - Multiple Stereotype
    - Restrict Application Of 19
- O -**
- ObjectType Enum
    - Automation Interface 188
  - Open Repository
    - Automation Interface Code Example 273
  - Override
    - Default Toolbox In Toolbox Profile 47
- P -**
- Package
    - Add And Manage, Automation Interface Code Example 274
    - Automation Interface 182
    - Automation Interface Repository 209
    - Profile 5
    - Save Profile 17
  - Parameter
    - Automation Interface, ElementFeatures Package 243
  - Partitions Collection
    - Automation Interface, ElementFeatures Package 244
  - Pattern
    - Add To MDG Technology 33
  - PDATA
    - Diagram Profile Attribute Values 51
    - Element Attribute In MiscData, Object Model 221
  - PI Macro
    - Code Template Syntax 104
  - Post-New Events
    - Add-In Model 142
    - EA\_OnPostNewAttribute 144
    - EA\_OnPostNewConnector 142
    - EA\_OnPostNewDiagramObject 143
    - EA\_OnPostNewElement 142
    - EA\_OnPostNewMethod 144
    - EA\_OnPostNewPackage 145
  - Predefined Tag Type
    - Assign To Stereotype 9
    - Define 9
  - Predefined Tagged Value Type
    - Filters 79
    - Reference Data 82
    - Structured 79
    - Syntax 79, 82
  - Pre-Deletion Events
    - Add-In Model 135
    - EA\_OnPreDeleteConnector 135
    - EA\_OnPreDeleteDiagram 136
    - EA\_OnPreDeleteElement 135
    - EA\_OnPreDeletePackage 136
  - Pre-New Events
    - Add-In Model 137
    - EA\_OnPreNewAttribute 139
    - EA\_OnPreNewConnector 138
    - EA\_OnPreNewDiagramObject 139
    - EA\_OnPreNewElement 137
    - EA\_OnPreNewMethod 140
    - EA\_OnPreNewPackage 141
  - Profile
    - Add Elements 6
    - Add Enumeration Elements 13
    - Add Metaclasses 6
    - Add Shape Script 14
    - Add Stereotypes 6
    - Add To MDG Technology 33
    - And Element Templates 3
    - Constraints 11
    - Create 5
    - Diagram, Create 50
    - Export 16
    - Import From XML 3
    - Package 5
    - Save From Diagram 17
    - Save From Package 17
    - Set Default Appearance Of Stereotype Objects 16
    - Stereotype 5, 22
    - Stereotypes 3
    - Tags 8
    - Tasks Pane, Create 52
    - Toolbox 45
    - Work With 5
  - Project Interface
    - Automation Interface 262
    - Project 262
  - ProjectIssues
    - Automation Interface Repository 212
  - ProjectResource
    - Automation Interface Repository 213
  - Properties
    - Automation Interface, ElementFeatures Package 245

- Properties, EASL
  - Action 111
  - Argument 111
  - Behavior 111
  - Call Event 111
  - ChangeEvent 111
  - Classifier 111
  - Condition 111
  - Construct 111
  - Edge 111
  - EventObject 111
  - Instance 111
  - Parameter 111
  - Primitive 111
  - PropertyObject 111
  - SignalEvent 111
  - State 111
  - StateMachine 111
  - TimeEvent 111
  - Transition 111
  - Trigger 111
  - Vertex 111
- Property
  - Automation Interface, ElementFeatures Package 245
- PropertyType
  - Automation Interface Repository 214
- PropType Enum
  - Automation Interface 188
- Q -**
- Query Methods
  - In Shape Scripts 70
- Quick Linker
  - Connector Names 27
  - Default Settings, Hide 26
  - Definition Format 23
  - Element Names 27
  - Example 25
  - Introduction 23
- R -**
- Re-entrancy
  - In Add-Ins 124
- Reference
  - Automation Interface 181
  - Automation Interface Repository 215
- Reference Data Tagged Value Type 82
- Reference Data Tagged Values
  - Create 83
- ReloadType Enumeration
  - Automation Interface 189
- Repository
  - Attributes 190
  - Author Collection 203
  - Automation Interface 190
  - Client Collection 204
  - Collection Class 205
  - Datatype 206
  - EventProperties 207
  - EventProperty 208
  - Methods 190
  - ModelWatcher 208
  - Open, Automation Interface Code Example 273
  - Package 209
  - Package, Automation Interface 189
  - ProjectIssues 212
  - ProjectResource 213
  - PropertyType 214
  - Reference 215
  - Stereotype 216
  - Task 216
  - Term 217
  - Use Extras, Automation Interface Code Example 280
- Requirement
  - Automation Interface, Element Package 230
- Reserved Names
  - In Shape Scripts, Connectors 74
  - In Shape Scripts, Elements 74
- Resource
  - Automation Interface, Element Package 231
- Risk
  - Automation Interface, Element Package 232
- RoleTag
  - Automation Interface, Connector Package 253
- RTF Report Template
  - Add To MDG Technology 42
- S -**
- Save
  - Profile From Diagram Context 17
  - Profile From Package Context 17
  - Tasks Pane Profile 56
- Scenario
  - Automation Interface, Element Package 232
- Script
  - Add To MDG Technology 41
- SDK

- SDK
  - Enterprise Architect 2
- Search
  - Add-In 126
- Search Data Parameter
  - Add-In Search 127
- Shape
  - <LabelID> 74
  - Attributes 64
  - Decoration 74
  - Editor 62
  - Label 74
  - Main 74
  - Source 74
  - Target 74
- Shape Attributes
  - Shape Scripts 64
- Shape Editor 62
- Shape Scripts
  - Add To Profile 14
  - Arithmetical Operations 74
  - Assign To Stereotype 59
  - Basic Shapes 75
  - Change Font Of Text 74
  - Cloud Path 75
  - Color Queries 70
  - Comments 74
  - Conditional Branching 70
  - Connector 75
  - Create 59
  - Custom Shapes 59
  - Display Element Properties 70
  - Double Line 75
  - Drawing Methods 66
  - Editable Field 75
  - Example Shape Scripts 75
  - Filled Arrow 75
  - Fonts 74
  - Getting Started 59
  - Introduction 59
  - Looping 74
  - Miscellaneous 74
  - Multiple Condition 75
  - Override Element Appearance 59
  - Properties, Connector 70
  - Properties, Element 70
  - Query Methods 70
  - Reserved Names, Connectors 74
  - Reserved Names, Elements 74
  - Return Statement 75
  - Shape Attributes 64
  - Shape Editor 62
  - Single Condition 75
  - Stereotypes 59
  - String Manipulation 74
  - Subshape 75
  - Subshape Layout 73
  - Syntax Grammar 63
  - Terminate Execution 74
  - Variable Declarations 74
  - Without Stereotypes 74
  - Writing Scripts 63
- Software Development Kit
  - Enterprise Architect 2
- Stereotype
  - Add Shape Script In Profile 14
  - Add To Profile 6
  - Add, Automation Interface Code Example 281
  - Automation Interface Repository 216
  - Custom 3
  - Define As Metatype 19
  - Dialog 3
  - Multiple, Restrict Application Of 19
  - Predefined Tag Types 9
  - Profile 22
  - Set Default Appearance Of Objects In Profile 16
  - Tagged Values In Profile 8
  - Tags For Supported Attributes 9
  - Tags, Define 8
- Structured Tagged Value Type 79
- Structured Tagged Values
  - Create 81
- StyleEx
  - Diagram Profile Attribute Values 51
- Submenu
  - Hidden, Create In Toolbox Profile 46
- Subshape
  - Example 73
  - In Shape Scripts 73
- Substitution
  - Conditional 88
  - Direct 88
  - Macro 70
- Supported Attribute
  - Create Composite Elements 21
  - Define Behavior On Creating Instance 20
  - Define Child Diagram Types 21
  - Metatype, In UML Profiles 18
  - Of Stereotype Tags 9
  - Stereotype, In UML Profiles 18
- Supported Stereotype Attribute Tags 9
- SwimlaneDef
  - Automation Interface, Diagram Package 260



Swimlanes  
 Automation Interface, Diagram Package 261,  
 262  
 Synchronization  
 Macros, Code Template Syntax 104  
 Syntax Grammar  
 Shape Scripts 63

## - T -

Tag  
 Profile 8  
 Tag Type  
 Predefined, Assign To Stereotype 9  
 Tagged Value  
 Connector, Use 10  
 Custom, Create 84  
 Element Package, Automation Interface 233  
 Macros, Code Template Syntax 100  
 Masked, Create 84  
 Reference Data, Create 83  
 Structured, Create 81  
 Tagged Value Type  
 Add To MDG Technology 37  
 Filters 79  
 Introduction 79  
 Predefined Reference Data 82  
 Predefined Structured 79  
 TaggedValue  
 Automation Interface, Element Package 233  
 Task  
 Automation Interface Repository 216  
 Task Panel  
 Add To MDG Technology 36  
 Tasks Pane  
 Allocate Toolbox To Contexts 55  
 Commands, Built In 53  
 Contexts, Define 55  
 Named Contexts 55  
 Profiles, Create 52  
 Run Add-In Functions From 54  
 Save Profile 56  
 Toolboxes, Define 53  
 Technology Event  
 EA\_OnInitialize\_Technologies 147  
 Technology Events  
 Add-In Model 146  
 EA\_OnDeleteTechnology 149  
 EA\_OnImportTechnology 149  
 EA\_OnPostActivateTechnology 148  
 EA\_OnPreActivateTechnology 147  
 EA\_OnPreDeleteTechnology 148

Template  
 Behavioral Model 107  
 Editor In SDK 116  
 Model, Incorporate In Technology 57  
 Term  
 Automation Interface Repository 217  
 Test  
 Automation Interface, Element Package 234  
 Toolbox  
 Add To MDG Technology 35  
 Connectors For Extending In Profile 49  
 Customize 45  
 Default, Override In Profile 47  
 Elements For Extending 48  
 Override Default In Toolbox Profile 47  
 Page Attributes 46  
 Profile, Create For MDG Technology 45  
 Profiles 45  
 Tasks Pane, Define 53  
 Toolbox Profile  
 Connectors For Extending 49  
 Create Hidden Submenu In 46  
 Items, Assign Icons For 47  
 Pages That Can be Overridden 48  
 Transitions Collection  
 Automation Interface, ElementFeatures Package  
 246  
 Tricks and Traps  
 Create Add-In 124

## - U -

UML Profile  
 And Element Templates 3  
 Create 5  
 Export 16  
 Import From XML 3  
 Save From Diagram 17  
 Save From Package 17  
 Stereotypes 3  
 Work With 5  
 Use Element Extras  
 Automation Interface Code Example 277  
 Use Repository Extras  
 Automation Interface Code Example 280

## - V -

Validation  
 Of Model, Configure For MDG Technology 56  
 Variable

## Variable

- Definitions, Code Template Syntax 115
- Definitions, Examples 115
- References, Code Template Syntax 115
- References, Examples 115

## VB

- Set Up In Automation Interface 178

## Visual Basic

- Connect To Automation Interface 176
- Set Up In Automation Interface 178

**- W -**

## Work With Attributes

- Automation Interface Code Example 282

## Work With Methods

- Automation Interface Code Example 282

**- X -**

## XMType Enum

- Automation Interface 189

Enterprise Architect Software Developers' Kit

[www.sparxsystems.com](http://www.sparxsystems.com)